An Investigation into the Planning, Organisation and Control of Rapid Software Development and Prototyping

Mark Andrew Stephens BSc (Hons)

A thesis submitted in partial fulfilment of the requirements of the University of Wolverhampton for the degree of Doctor of Philosophy

June 1999

Abstract

Software development involving prototyping has been widely written about since the early 1980s. This project has been carried out with the intention of investigating the various claims made about it. The first section of the research examines the background to software development and the emergence of alternative approaches to carrying it out. It considers the options open to developers and then examines the background and application of prototyping. It reviews the published work (including survey evidence) on the management and control of prototyping and the ways that this can be carried out effectively. The second part of the work evaluates the results of a series of studies that have been carried out. These comprise several developments for systems built for a metal finishing company and also the analysis of using function points measurement in the development of a system for a local authority, in each study prototyping was used as part or all of the development,. The final part of the work proposes a model for controlling the prototyping process. The second part is a quantitative investigation in to the effectiveness of prototyping as part of a RAD style development. The report concludes with overall recommendations and scope for further research.

Table of Contents

Appendices

Appendix 1.
Background to developing an information system for a metal finishing company.

Appendix 2a.
Draft specification for task to be carried out by groups

Appendix 2b.
Data for field experiment

Appendix 3. Questionnaire used in Chapter 4.

Figures

Graphs

Chapter 4

Chapter 8

Chapter 10

Tables

Acknowledgements

I would like to thank all the people who have contributed towards the work carried out in this project, in particular: -

Peter Bates and John Cooper of the University of Wolverhampton for their advice, encouragement and considerable patience throughout the time that this investigation has been underway.

Andrew Gilson, formerly of Frosts Electroplating and various other staff at the company for their commitment to the work carried out and their invaluable time spent evaluating prototypes.

Wayne Clark for demonstrating that research work is not impossible and keeping my spirits up in times of extreme doubt.

Linda and my parents for putting up with me!

It is dedicated to John and David.

Part 1. Prototyping and Rapid Software Development A Review of Progress, Problems and Management.

The first part of the document considers the background to problems in software development and the emergence of possible ways of combating some of the problems that need to be addressed.

The aims and objectives of the research are documented together with the research methods used (and the reasons for using them).

The main part of the first section is a review of the published literature in the subject area. It includes a review of several surveys that have been carried out in order to try and ascertain whether there are any consistent threads that run through these that can provide useful information to developers considering the use of prototyping and rapid application development.

The final section of the first part considers the issues of management and control of the prototyping process in more detail. These points are often cited in the literature as being areas of particular concern. Hence they are examined in some depth to try and evaluate those mechanisms that currently exist and how they can be used to attempt to help organise the management and control processes within prototyping and RAD.

Chapter 1. Introduction

1.1 Project background

Considerable evidence is available in published literature that prototyping is becoming an increasingly important part of systems development. This is despite the fact that there is also evidence that many systems are still being built in the same way that was common in the late 1960s. I.e. little real account being taken of users' actual requirements and with technical computing issues prevailing over system usability and convenience. Although various structured and formal methods have become more widespread there still seems to be an inability amongst software development professionals to deliver the correct product to the satisfaction of clients and users on time.

Software can be built using a wide variety of development methods. These range from highly prescriptive and monolithic "cookbook" approaches to "softer" techniques with less formal structure associated with them. However as business organisations seek to find a competitive edge, there is a need to build more complex systems in shorter time scales and at less cost. Crinnion (1992a) states that:

> "It is now generally accepted that in order to be successful, a modern business must be highly flexible, and able to adjust its procedures to respond quickly to a volatile environment."

He suggests that traditional development methods that are based on a sequential life-cycle cannot cope with this type of environment.

This view is also shared by Ince (1991) who observes that:

> "....the realisation that the software project is not the clean, phase-oriented process that text books have been describing for a number of years. Software projects tend to be extremely messy. The reason for this is change."

Prototyping has existed as a form of software development since the early 1980s. It has been extensively documented in both commercial and academic journals since that time, but there still appears to be some reluctance amongst commercial developers to embrace it enthusiastically as a practical software development method.

This research project is concerned with examining approaches using prototyping that can be taken by software developers in an attempt to deliver quality software on time and within budget, features that still create many problems within the software industry! An example is the recent failure of a $165m car rental and hotel reservation system (Bicknell 1995), the causes of problems included an incomplete statement of requirements, lack of user involvement and constantly changing requirements and specifications. Typically these are all factors that prototyping in its various forms attempts to address.

Given that the outcome above is not atypical of software development, the work within this research project attempts to show how prototyping can be successfully used in software development. It illustrates through a review of the literature, survey evidence, action research and experimentation what prototyping can and cannot achieve.

This project was in some ways inspired by the work carried out and first published by Mayhew in 1987. In the concluding remarks of that work it is noted that further research requires to be carried out in information systems prototyping, particularly in the following areas:

- Planning for prototyping
- Organisation of prototyping
- Control of prototyping
- Tools for prototyping

In 1992 the IEE held a one day seminar on Software Prototyping and Evolutionary development. In the keynote address (Ince 1992) it was noted that the main problems existing in prototyping were those to do with management and quality. Included in this were the difficulties of monitoring and controlling the process and trying to ensure that systems unlikely to be feasible were not produced.

It is interesting to note that in the five years that elapsed between the two dates it seems that many of the problems remained the same. Hence it is these issues that form the central theme of the study.

Although prototyping has now existed for around fifteen years in a recognisable form it still is perhaps considered to be a subject on the periphery of mainstream computing. However there is some evidence that prototyping has evolved in to Rapid Application Development (RAD) and that this approach will be more widely adopted in the software industry.

RAD has now been used since the early 1990s (E.g. Martin 1991, Kerr and Hunter 1994) as an approach to software development which takes into account human factors and corporate climate as well as technology (Baum 1992, Gill 1992). RAD is seen to be increasingly important as organisations undergo Business Process Re-engineering (BPR). Frost (1994) notes that BPR is often enabled by information technology and hence the technology itself must be more responsive.

Companies developing increasingly complex applications are required to produce software in shorter time scales. They must also assimilate change during development and produce systems which are both flexible and adaptable after implementation, in order to maintain a competitive advantage. An important element of RAD is the prototyping process, but it is not clear how far prototyping has been accepted by software developers and what methods are used to manage it successfully.

Rapid Application Development (RAD) with the extensive use of prototyping has rekindled the idea of highly user centred computing solutions. It is the planning, control and management of these approaches that this research is primarily concerned with.

The research attempts to examine the many claims for prototyping in order to try and find out where real benefits exist, but also where problems are likely to lurk. By critically evaluating these points it is hoped that developers will appreciate both what prototyping can help to achieve, but also where other mechanisms are required to ensure the delivery of quality software within budget.

1.2 Research aims and objectives

Aims

The aims of this research project are to try to discover whether the use of prototyping and rapid software development can enhance the software development process.

- As part of the literature review one of the initial aims of the research was evaluate the similarities and differences between the various surveys carried out in the subject area and to discover whether any particular recurring themes appeared in the results reported by them. In order to carry this out it was decided to examine the published survey evidence that had been undertaken in both in the United Kingdom and North America.

- An investigation in to the effectiveness of prototyping techniques and an examination of the usefulness of control mechanisms that could be applied was undertaken. Hence an exercise in action research where several pieces of software were written using a variety of prototyping techniques in a commercial development environment was carried out.

- Research in to the applicability of function point analysis to discover how useful this particular technique could be in helping to control projects involving a high degree of prototyping was undertaken. The results of a rapid software development project where function points had been used as part of the project management process were analysed in order to investigate this technique.

- The research aimed to discover how a rapid software development project would affect the production of software in terms of its overall functionality and quality and how the use of prototyping in particular affected the development process. Hence an experimental study where software development teams produced working implementations of a system specification was set up. The teams could use a model of systems development that is proposed in this project.

Objectives

The overall objectives of the research project are to assess the value of rapid software development and prototyping.

The four aims listed on the previous page were formulated in order to try and find out whether: recurring themes about the use of prototyping occurred in the survey literature; how useful prototyping can be in conjunction with various control mechanisms; if function point analysis and prototyping can be used in conjunction to provide useful project management information and finally to discover what influence rapid software development and prototyping have on the development of software.

1.3 Research methods

Researching problems associated with application software development and in particular information systems software can be a difficult area of study. It is important to appreciate that choosing the appropriate methods of research is itself an area of debate and that careful thought must be applied to ensure that sensible methods are adopted. This difficulty is well illustrated by the technical correspondence that appeared in the Communications of the ACM (Galliers and Land 1987, Jarvenpaa 1988).

It has been argued (e.g. Hirschheim in Galliers 1994)[1] that information systems have all the difficulties associated with social sciences because they are largely human or social in nature. If this assertion is taken as true then scientific research methods where reductionism, repeatability and refutability are essential may not be possible to use accurately. Hence in order to plan the research project it is necessary to try and choose those methods which are going to give the greatest help in the particular domain that is being investigated.

The table reproduced overleaf summarises a range of methods that can be adopted for research in this area. It lists the main features of the particular method, together with the perceived strengths and weaknesses.

[1] This essay gives a good historical overview and introduction to contemporary thinking in information systems research.

Table 1.1 A summary of the key features in alternative approaches to information systems research (summarised from Galliers in Galliers 1994)[2].

Approach	Key features	Strengths	Weaknesses
Laboratory experiments	Identification of precise relationships between chosen variables via a designed laboratory situation, using quantitative analytical techniques, with a view to making generalizable statements applicable to real-life situations.	The solution and control of a small number of variables which may then be studied intensively.	The limited extent to which identified relationships exist in the real world due to oversimplification of the experimental situation and the isolation of such situations from most of the variables that are found in the real world.
Field experiments	Extension of laboratory experiments into the real-life situations of organisations and/or society.	Greater realism; less artificial/sanitized than the laboratory situation.	Finding organisations prepared to be experimented on. Achieving sufficient control to enable replication, with only the study variables being altered.
Surveys	Obtaining snap shots of practices, situations or views at a particular point in time (via questionnaires or interviews) from which inferences are made (using quantitative analytical techniques) regarding the relationships that exist in the past, present and future.	Greater number of variables may be studied than in the case of experimental approaches. Description of real world situations. More easy/appropriate generalisations.	Likely that little insight obtained re. The causes/processes behind the phenomena being studied. Possible bias in respondents (cf. Self-selecting nature of questionnaire respondents); the researcher, and the moment in time which the research is undertaken.
Case studies	An attempt at describing the relationships which exist in reality, usually within a single organization or organizational grouping.	Capturing 'reality' in greater detail and analysing more variables than is possible using any of the above approaches.	Restriction to a single event/organization. Difficulty in generalizing, given problems of acquiring similar data from a statistically meaningful number of cases. Lack of control of variables. Different interpretations of events by individual researchers/stakeholders.

[2] This table is explained in greater detail in the cited reference.

Forecasting, futures research	Use of such techniques as regression analysis and time series analysis, or the Delphi method and change analysis, to extrapolate/deduce likely/future possible events or impacts.	Provision of insights into likely future occurrences in situations where existing relationships may not hold true in the future.	Complexity and changing relationship of variables under study. Lack of real knowledge of future events. Scenarios are not 'true' pictures of the future but enable decisions re. reactions in different 'futures'. Dependent on precision/relevance of past data and expertise of scenario builders. Possibility of self-fulfilling prophecies.
Simulation, game/role playing	An attempt at copying the behaviour of a system which would otherwise be difficult to/impossible to solve analytically, by the generation/introduction of random variables.	Provision of an opportunity to study situations that might otherwise be impossible to analyse.	Similar to experimental research in regard to the difficulties associated with devising a simulation that accurately reflects the real world situation.
Subjective, argumentative (cf. Phenomenology, hermeneutics	Creative research based more on opinion/speculation than observation, thereby placing greater emphasis on the role/perspective of the researcher. Can be applied to existing body of knowledge (reviews) as well as to actual/past events/situations.	Useful in building theory that can subsequently be tested. Creation of new ideas and insights. Recognition that the researcher will interpret what is being studied in a particular way. Contributes to cumulative knowledge.	Unstructured, subjective nature of research process. Despite making the prejudice of the researcher known, there is still the likelihood of biased interpretations, a problem which is compounded by the time at which the research is undertaken.
Action research	Applied research where there is an attempt to obtain results of practical value to groups with whom the researcher is allied, while at the same time adding to theoretical knowledge.	Practical as well as theoretical outcomes most often aimed at emancipatory outcomes. Biases of researcher made known.	Similar to case study research, but additionally places a considerable responsibility on the researcher when objectives are at odds with other groupings. The ethics of the particular research are a key issue.

The research methods chosen to carry out this particular project were selected in order to try and satisfy several different criteria.

In particular the author agrees with Kirk and Miller (1986) that:

> "qualitative research conducted as science should complement nonqualitative research".

Hence it is important to choose a method or methods that allows this to take place. This research project maps closely on to the model proposed in Galliers and Land (op.cit). It differs only in that the theory building exercise is carried out after the case study/action research section.

Research Question

Survey Research

Theory Building

Case Study/Action Research

Theory Testing in a Field

Theory Extension

Figure 1.1. Use of Alternative Research Methodologies (Jarvenpaa op.cit)

Hence the research work used four techniques during the project. They are listed below:

- Surveys

It was decided that despite the possible problems of self selection cited in some literature that a "survey of surveys" would provide a useful insight into the application of prototyping and rapid application development. By comparing a specially written survey with comparable other studies, it was hoped to gain a good overview of particular aspects of the subject. This work is documented in Chapter 4, which is the final part of the literature review documentation.

- Action Research

The opportunity arose to build application software for a commercial development in order to solve a non-trivial computing problem. It was not possible to treat this as an experiment as controlled conditions and hence repeatability would have been extremely difficult to set up and manage. According to the definitions given in Table 1.1, it was run as an exercise in action research rather than as a true case study as the researcher played an active role in nearly all parts of the projects undertaken and hence their presence would inevitably affect the outcome of the research. This work is documented in part two of the report in Chapters 6 and 7.

A further project was undertaken to discover more about the use of function point analysis and prototyping. This part of the research was also carried out as an action research exercise in that development work was undertaken by one researcher, raw data collection by another, and finally data analysis by a third individual. The results of this exercise are recorded in Chapter 8.

Both these types of approach enable particular problems to be studied in considerable depth, however in both of these cases it is acknowledged that the findings are only really applicable to the project at the time of the research and hence generalisation of the results will certainly have no quantitative significance. Nonetheless such studies do add to the published literature in the area.

- Laboratory and field experiments

For the final part of the research project an attempt to use a more scientific method for investigation was undertaken. It is difficult to decide whether the study undertaken is a laboratory or field investigation as it exhibits certain characteristics of both. It was hoped to control certain parameters in the project (as in the case of a laboratory experiment), but it was also the intention to try and replicate a real life scenario which had more in common with a field experiment. It would probably be accurate to say that the study represented a hybrid. Hence although it is possible to use quantitative analytical techniques, it is also acknowledged that certain issues of replication would be more difficult to achieve than in a true laboratory experiment. This typifies the dilemma in information systems research, i.e. if more freedom is given to participants in the study (and hence realism is increased), this can only be at the expense of scientific and quantitative repeatability and accuracy.

Nonetheless it was decided that a part of the research should use methods more strongly associated with a scientific approach as it should allow quantitative evaluation to take place.

By using a variety of research techniques it was hoped to be able to carry out a thorough investigation in the area of rapid software development and prototyping and to develop mechanisms that allowed it to be organised and controlled in an effective manner.

Chapter 2. Software development methods - A background and overview

This section reviews the main forms of software development processes that have been used. It considers the issues that have led to the emergence of Rapid Application Development (RAD) style methods.

In 1983 William Davis wrote "Systems Analysis and Design". This type of publication typified the literature of the era which encouraged software students and practitioners to move into structured techniques. It was perceived by software developers that this form of development could help move software out of the chaos of traditional ad hoc methods into an age where software could be designed, constructed and maintained in a more effective manner.

Perhaps the most significant feature of this book is embodied in the very first diagram which is reproduced below.

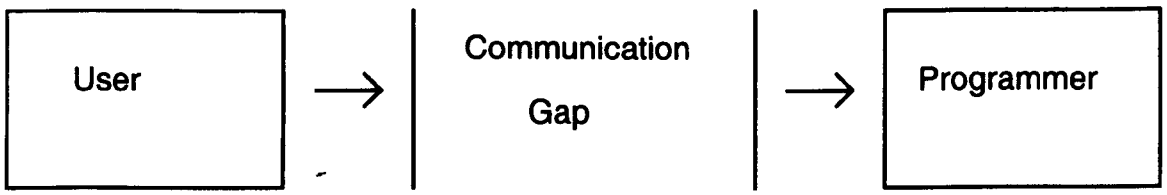


Figure 2.1. A communication gap separates the user from programmer (From Davies 1983)

Davis states:

> "The users knows the problem but can't solve it. The programmers might be able to solve the problem, if only they understood the problem. Complicating matters is a communication gap (see Figure 2.1): at times, programmers and users seem almost to speak a different language."

It was also around this time that prototyping was becoming a subject of interest within computing. It is interesting to note that in one of the early papers published on the subject (Mason and Carey 1983) it was stated that:

> ".....prototyping techniques have been widely and effectively employed in the development of information systems. This appears to be in part because prototyping approaches overcome fundamental problems between developers and users."

Unfortunately in the 1990s the communication gap as described by Davis seems to be as large or even larger than ever. The "wide and effective employment of prototyping" does not seem to be the case. Perhaps development involving prototyping if planned and organised in an effective manner could prove to be the solution that helps to bridge this gap?

2.1 Software development methods

The planning, design, development, coding and maintenance of software for computer systems of any significant size, can be carried out by using a variety of different techniques. Project managers have a choice of various development methods open to them before commencing on a software project. Boehm *et al* (1984) classified these into three distinct categories:

- Building and Fixing,
- Specifying,
- Prototyping.

It could also be argued that since the advent and widespread acceptance of the micro computer that a significant amount of activity now falls under the heading of :

- User Developed Applications (UDA).

and that in addition to Boehm's categorisations, there has been an extra category of:

- Rapid Application Development (RAD)

The final heading has many characteristics in common with prototyping so should perhaps be considered to be an extension of that alone, although this is not a universally held view. It is considered in more detail later in the work.

Each of these approaches will be considered briefly.

2.1.1 Building and fixing

The earliest form of system building was usually carried out using a building and fixing approach. Programs were coded with little forward planning and with scant regard for future changes or maintenance requirements.

Unfortunately the very nature of building and fixing with the emphasis on error correction as "fixing" at a late stage makes this type of development potentially very error prone and hence expensive. This method does not have a formal plan so that errors often occur because of unforeseen events arising. The development concentrates heavily on writing code from a very early point in the process and little emphasis is placed on trying to discover and exclude errors prior to code being written. This form of development is likely to require frequent modification throughout its construction as the requirements are unlikely to have been fully discovered during the initial stages of the project. In addition, the costs of maintaining systems developed in this manner are often very high. The lack of coherent structure in the software means that any changes that are necessary become time consuming and difficult to implement.

Boehm (1981) suggests that errors made in the early stages of a software project are typically from one hundred to one thousand times more expensive to correct if they remain undiscovered until the system is in the final stages of programming or has been implemented.

Towards the end of the 1960s people began to talk of the "software crisis." This term was coined because of the drawbacks that large systems seemed to have associated with them. Often these systems:

- Took too long to build
- Cost too much money
- Could not evolve
- Did not provide the functionality required of them.
- Were built, but never used.

Glass (1977) gives details of the failure of several large computing projects as evidence of the scale of this "crisis". More recently, well publicised events involving partial or even total failure of large software systems demonstrate how expensive, damaging, and time consuming error rectification can be. The examples of the AT&T telephone system problems in the U.S.A in January 1990 as reported in the Guardian 25th January 1990, and the complete abandonment of the London Stock Exchange Taurus trading system both illustrate this well.

Evidence from a variety of sources (e.g. Lientz and Swanson 1980) suggests that maintenance costs can be responsible for up to seventy percent of the total project cost of a software system. Lehman (1980) states that the need for continuing change is intrinsic in computer software usage and that as a result programs written must be made alterable in order to try to keep unit maintenance costs to a low level. It is important to appreciate that any form of development that ignores potential maintenance or change is likely to end up as a very expensive project. The rise in popularity of alternative forms of software development and of tools designed to re-engineer existing code (which is becoming increasingly hard to maintain), suggests that ad hoc methods of producing software with little regard to specification or to future changes in requirements are seen as increasingly impractical by more enlightened project managers.

Although the initial appeal of a fast start to creating code may appeal, the long term consequences are likely to be expensive.

2.1.2 The specifying approach

The advent of software being constructed in the manner of an engineering exercise has brought about a change in attitude regarding the development of software. The idea that software development should be treated as a science rather than an art has become increasingly widespread. Lamb (1988) states that software engineering is the application of principles of existing engineering disciplines to try to overcome the problems of the software crisis. This approach has been documented widely in the literature (e.g. Sommerville 1995 and Pressman 1992).

The principles of engineering are applied to the development process. Design methods which bring formality have been adopted by developers who wish to use a more systematic method of writing software. This manner of managing and controlling software writing is known as the specifying approach.

The specifying approach involves breaking the development of a system into sequential stages. Typically (though depending on the particular method used,) the system requirements phase will be followed by a design phase, then coding, implementation, and testing followed by maintenance. This type of development is well documented in the literature and is available in different styles, e.g. SSADM, Yourdon Structured Design and Information Engineering.

Building computer systems is a complex and time consuming process. Attempts have been made to depict the process as a series of discreet stages. This has been represented by various methods. Perhaps the most widespread has come to be known as the software life-cycle model. One of the first proposed was by Royce (1970). Since then the life cycle model has had refinements and alterations suggested. Figure 2.2 shows a typical depiction of the life-cycle model.

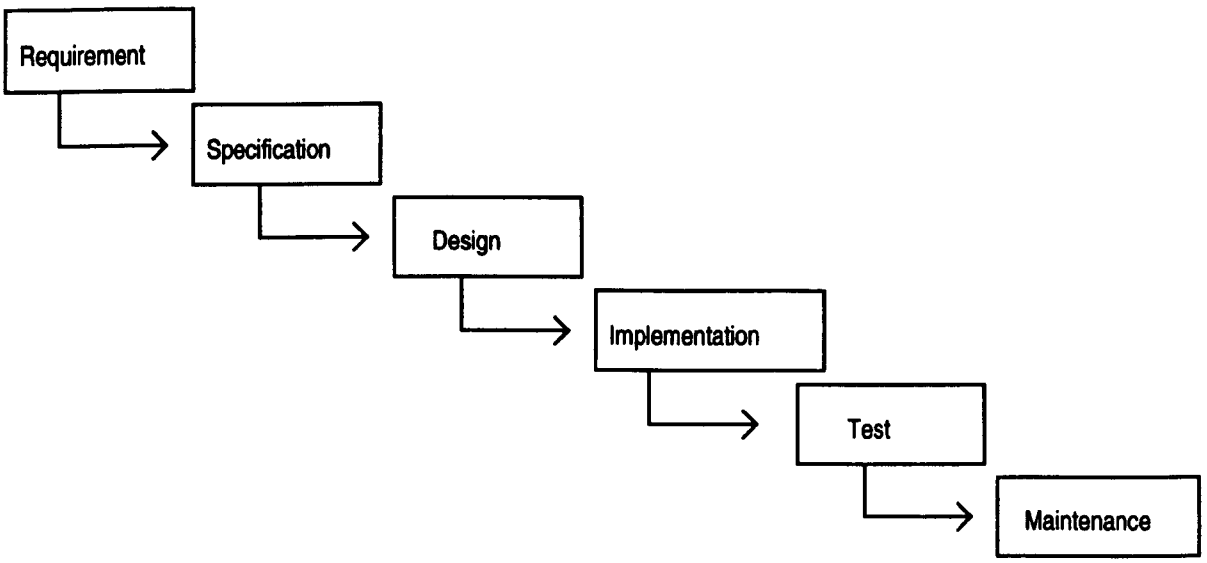


Figure 2.2. A typical representation of the waterfall model

From the figure it can be seen that the information flows forwards from one stage to the next in the development process. However there is also a backward flow which updates previous stages with new or changed information from a later point in the development.

By the late 1980s this type of method had certainly become widespread with many different forms. The National Computer Centre (NCC 1988) give summaries of the attributes of a variety of design methods that fall within this type of software development approach. Examples include SSADM, YOURDON and JSD. Each is outlined overleaf.

- SSADM (Structured System Analysis and Design Method) uses three phases of activity. Feasibility study, systems analysis and systems design. Each step contains a series of tasks which may be represented as a task list in both text and diagrams. Each element should have a clearly defined start and finishing point.

- YOURDON developed by Edward Yourdon models feasibility, structured analysis and structured design. It uses data flow diagrams, process specifications, entity relationship diagrams, data dictionaries and prototypes to develop a written and graphical diagram.

- JSD (Jackson Systems Development) starts with a statement of need and consists of three main phases, modelling (using diagrams to represent the subject area as processes); function (where the outputs are established); and implementation (the specification is made compatible with the target environment).

More recent approaches also include methods to build systems using object oriented techniques. As with the previous design paradigms a variety of alternative methods are available. (E.g. Booch 1994, Cook and Daniels 1994).

These are by no means the only methods available but serve to illustrate the pattern that structured design methods have. Usually the stages in the system development are broken by reviews, discussions and the signing of various documents which represent parts of the proposed system. The implementation is built up in sequential stages with one phase finishing prior to the start of the next part.

Unfortunately most of the methods of analysis and presentation rely heavily on technical diagrams and specially devised notations which users may find difficult to understand. This has been considered a problem for some time, indeed Floyd (1984) noted in a case study that:

> "The users were not able to read the formal specifications - data flow diagrams, data definitions, data structure diagrams - produced by the project."

Data Flow diagrams (e.g. Martin 1986), data dictionaries (e.g. DeMarco 1979), and entity life histories (e.g. Jackson 1983) are typical examples of diagrams or techniques that are used to represent parts of a problem in terms of a particular design method.

Figures 2.3 and 2.4 are examples of the type of representation that are used by these methods.

Users are nonetheless expected to visualise how such a system is likely to perform and compare this with their original requirements. Whether this can be satisfactorily achieved is open to contention. Brooks (1987) states:

> "Much of present-day software acquisition-procedure rests upon the assumption that one can specify a satisfactory system in advance, get bids for its construction, have it built, and install it. I think this assumption is fundamentally wrong".

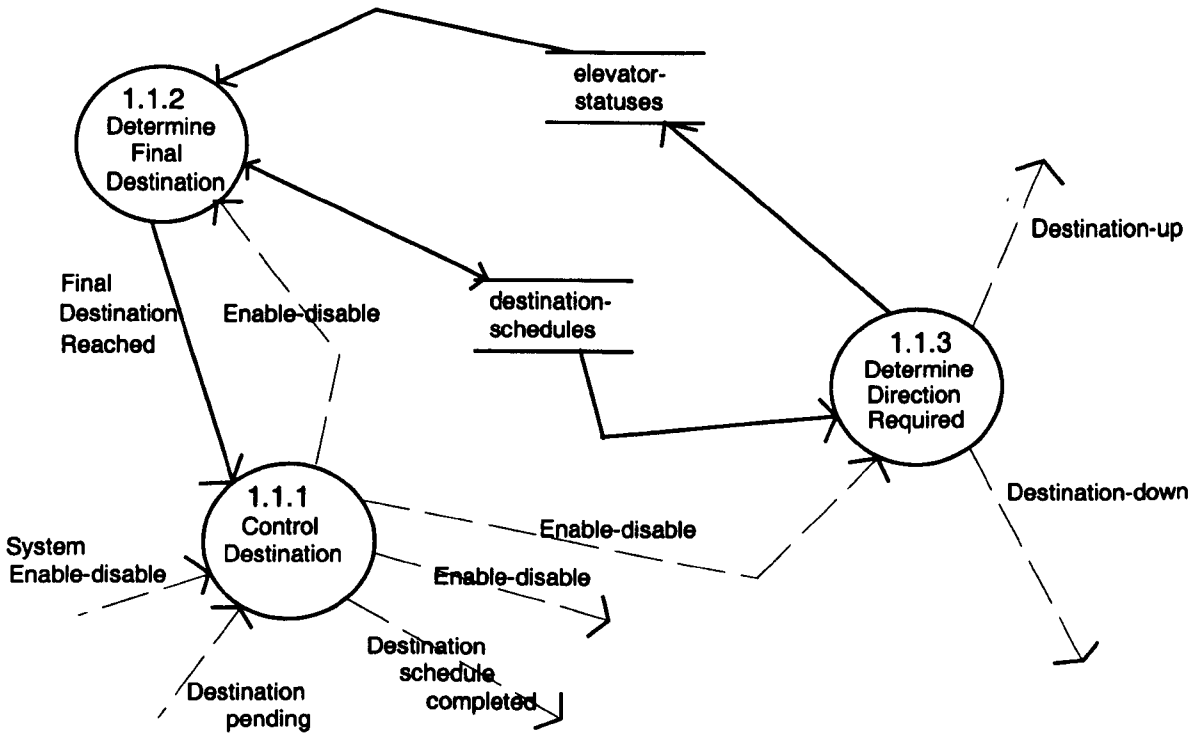


Figure 2.3. A data flow diagram showing a set of sub-processes in a lift control system (Adapted from Yourdon 1989)

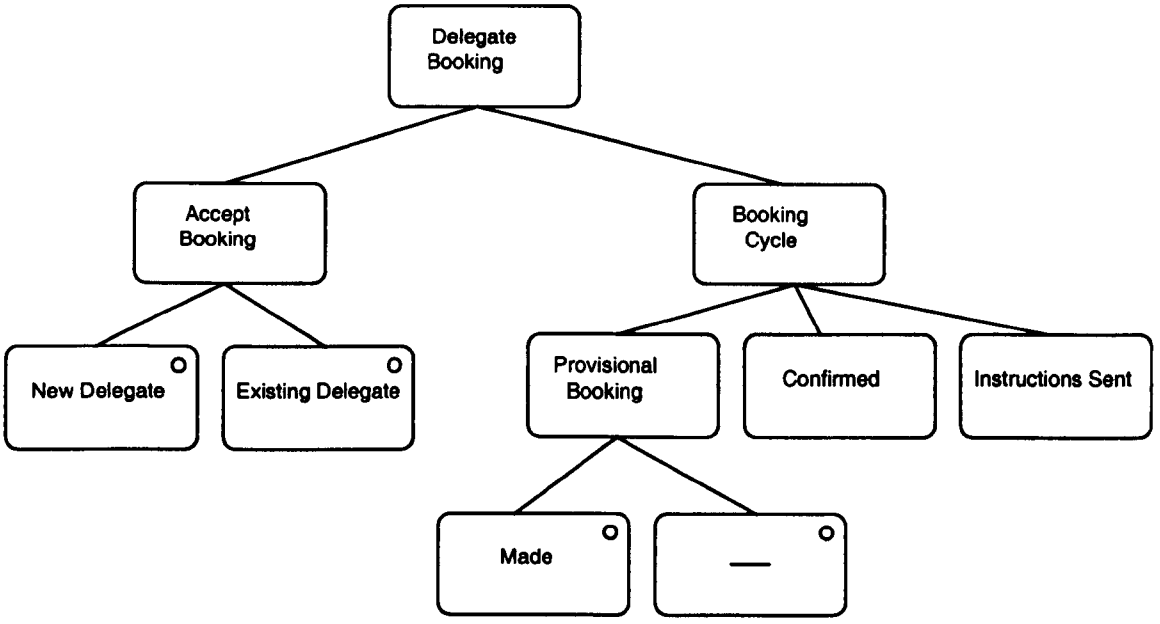


Figure 2.4. A provisional Entity-Life-History for a booking system (Adapted from Eva 1992)

Luqi (1989) reinforces this idea, she states that the design of accurate and stable requirements cannot be completed until users gain some experience with the proposed software system. Users find formal methods of representing design solutions hard to understand and cannot relate them to their own experience (Harker 1988). This effectively locks them out of the design process and does not allow their skill or knowledge to be utilised.

Typically a sequential design method will only allow a working part of the system to be examined at a comparatively late point in the project. Users are then required to validate the system in terms of the previously written specification. Boehm and Papaccio (1988) point out that software rework costs are still high when using this type of method as there is still a tendency for errors of requirements and specification to be discovered at a late stage in the project.

Where developers have good experience of the type of system under development and the users are very clear about what they want, a linear strategy is likely to produce good results (Brittan 1980). However, imprecise initial requirements, or an underestimation of the complexity of the problem is likely to lead to a limited or incomplete solution only. In such cases it is likely that a sequential design strategy will lead to problems at the later stages of the development. Gould and Lewis (1985) suggest that:

> "Experience shows that iterative design should not be thought of as a luxury tuning method that puts finishing touches on a design (at great expense)."

The system installed as the "final" version represents a set of agreed conditions at a particular point in time. Any modifications required which are not directly attributable to obvious software shortcoming will tend to be regarded as additions to the original specification and hence extras. The system is not regarded as dynamic and so problems are likely to arise for the reasons stated earlier. The responsibility for bearing the cost of enhancing this type of system is likely to be a problem if ambiguities or misunderstandings have been propagated through the design process and into the final installed software.

Lehman (1989) notes that:

> "A computer system being used to address a specific application can rarely if ever be frozen."

and additionally,

> "The application domain is also changing independently."

Pilote (1988) suggests that experience has taught computer professionals that success in a software project requires the minimum amount of change, particularly in their requirements. Hence it is not in the interest of developers to create problems by encouraging users to add, amend or in any way change their wants after the first phase of a project.

From the above it can be seen that the use of structured design and the specification it leads to, attempts to impose an ordered and sequential or hierarchical set of processes on software development. To make this possible certain assumptions have to be made. These suppose that a user can correctly specify a system and then validate the specification using text and a set of diagrams representing the system in an abstract form. Whether this process can be adequately carried out is not universally accepted. The difficulties inherent in this approach were recognised by the USA Defence Science Board Task Force on Military Software 1987 (See Balzer 1987) who observed that:

> "Current software systems are so complex that even the most diligent thought process cannot envisage them precisely and thoroughly enough that a correct and appropriate specification can be produced."

However the DoD still maintained a standard approach to system development in 1994, this was known by as DoD-STD 2167A. By combining the requirements of this standard with a typical research and development approach such programs typically take from five to eight years from initial concept through to implementation. (Cross and Estrada 1994).

2.1.3 Prototyping

Prototyping is a form of development which aims to reduce the incidence of errors and increase the acceptability of a system to the users. It has come about as a result of fundamental problems that exist with the specifying approach. The life-cycle model (as stated above) assumes a linear development cycle. Ralston (1988) gives three underlying assumptions that the traditional life-cycle model makes:

- A complete understanding of the systems requirements can be derived at the start of a project.

- It is possible to define discrete steps which can be separately verified before continuing to the next stage.

- The outputs of each stage can be traced back to the original requirements.

He states that these assumptions are almost inevitably incorrect because of the iterative nature of any large software project. The proponents of prototyping acknowledge the necessity of iteration and encourage it as a part of the systems design process. Brown *et al* (1988) state that very few designs for major systems have been successful without an extensive redesign.

Prototyping involves an iterative approach to building software, a typical model is shown in figure 2.5.

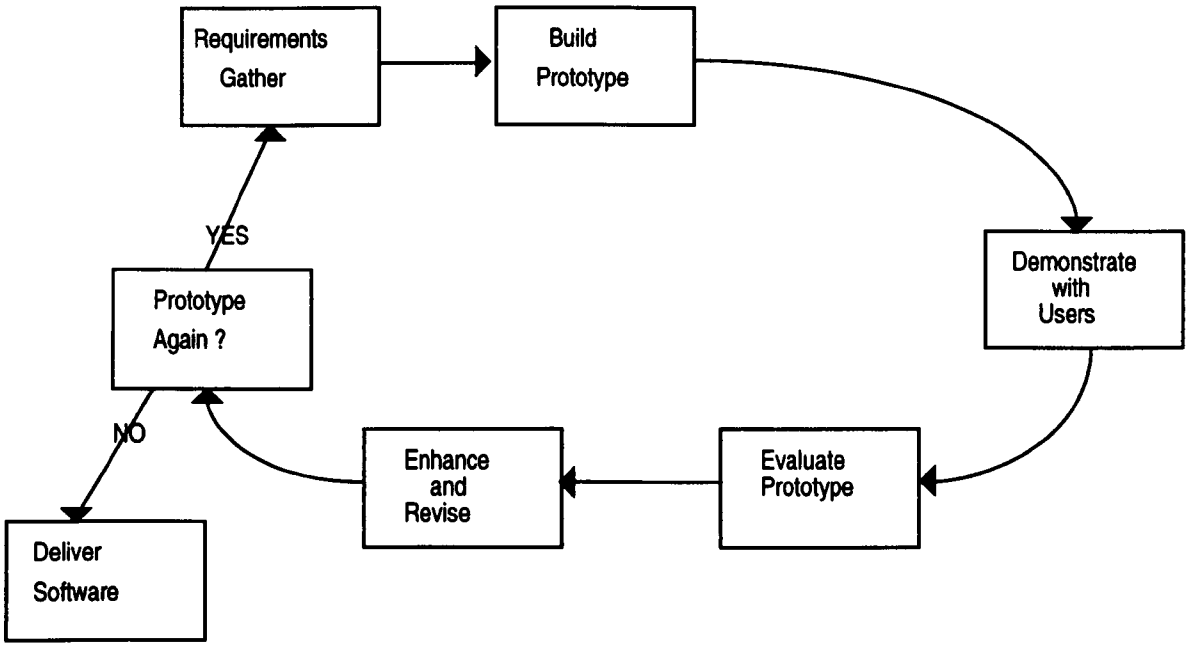


Figure 2.5. A typical representation of the prototyping process

Boar (1984) defines prototyping as a method for extracting, presenting, and refining a users needs by building a working system. The use of prototyping also aims to develop systems in significantly less time than alternative techniques (Gray *et al* 1988). Although there is not a lot of controlled empirical evidence to back up these claims some experimental results support the time and cost savings involved. An experiment which monitored the development of a software system using both specification and prototyping approaches (Boehm *et al* 1984) produced results that strongly favour the latter method. The study concluded that:

- The teams using prototyping produced products that were on average 40% smaller and took 45% less effort to complete.

- Peer group evaluation rated the likely maintainability of the prototyped products as significantly better.

- The prototyped products were easier to learn and use, and had better human-machine interfaces.

- The workload running up to the deadline was reduced in the teams using prototyping.

The use of prototyping where working mock-ups of part of a system are built may help eliminate errors at the beginning rather than towards the end of the systems

development. Users can see how parts of a system are likely to work and can comment on their usefulness much earlier on in the development process.

During the research project, prototyping was used extensively for a variety of tasks. A more in depth review of the subject is discussed in Chapter 3.

Although prototyping appears to provide a solution to some of the problems that continue to exist if structured methods are used, it is important to be aware of pitfalls that may befall the unwary developer. Necco *et al* (1989) list four points that can create problems when using prototyping. These are:

- Lack of standard procedures
- Use of prototype as a production system
- Lack of suitable tools
- Inadequate systems analysis. Prototyping can lead to the wrong problem being focused on.

Dickerson *et al* (1988) also suggest that there are potential problems that exist when using the prototyping approach. They note that:

> "One unexpected 'problem' with rapid prototyping is the fact that the prototype can look like a real product. If delivered as an interim system more feedback and a better system might ensue, however if it is too slow, does not have the required functionality and crashes frequently then the customer is likely to develop a negative attitude to the final system when it is delivered."

Iivari and Karjalainen (1989) noted that prototypes in an experiment proved to be highly effective for the early identification of faults, but also suggest that prototypes can lead to unrealistically high expectations and that considerable emphasis must be placed on specifying the idea that is being prototyped.

As prototyping has become more widely written about the number of definitions of what it actually represents has substantially increased. It has even got to the stage that very different procedures within the same organisation are described by the same term

(Crinnion 1992b). However he states that there are now two generally accepted forms and that they are :

- Throw-away
- Incremental (or evolutionary)

The former is merely a quickly built piece of software produced to identify basic requirements and is discarded once it has been used. The latter involves more rigour and structure in that the final prototype evolves into the first operational version of the released software. These definitions now seem to have become the *de facto* standards, although alternatives are still discussed.

2.1.4 User developed applications (UDA)

In many organisations the overall responsibility for designing and building application systems no longer rests solely with the central Information Systems (IS) departments. There has been an accelerating trend towards end-users taking over the task of developing systems themselves. Galletta and Hufnagel (1992) describe aspects of this form of development as:

> "the direct assumption of system development and processing tasks by the user.....for his own direct benefit".

Generally this form of development is described as a subset of the larger area known as End User Computing (EUC). Depending on the tasks involved the term EUC normally involves the array of activities associated with the employment of organisational computer resources outside the IS departmental control, whereas UDA is the development and use of business applications by end-users with little or no formal IS training.

There are drawbacks to this form of development which are widely documented in the literature (E.g. Maglitta and Hildebrand 1993, Karten 1991). Typically the main problems are concerned with the lack of rigour regarding the manner in which data is stored and manipulated, the lack of sophistication of the user interfaces, and the inadequate level of documentation. These problems become worse when the designer of the system leaves the organisation and responsibility for maintenance is passed onto other individuals.

However UDA is an important aspect of modern business computing as it can be seen as providing "user friendly software" (Rockart and Flannery 1983) which can allow business users to move towards greater control of their own computing needs.

The emergence of this form of development is attributable to a combination of several different reasons.

The decreasing cost of computing hardware has led to the widespread use of PCs within all aspects of modern business, hence there is now a much wider range of people using computing compared to 10 or 15 years ago. In addition these machines are often equipped with 4GL and similar software that allows users to perform tasks that were previously carried out by the central IS department. (Lee *et al* 1991). Using a 4GL as an interface to an application generator and being non-procedural can allow users to specify "what to do" rather than "how to do it". In addition, many of the current application generators can be easily integrated with a database system which allows easy data definition and manipulation. The incorporation of report writing facilities, business graphics, screen painting aids and structured querying languages enable user-developers to produce applications ranging from simple queries and graphics to building complete information systems. (Cotterman *et al* 1989, Lin 1990)

The spread of communications technology and the interconnectivity of different systems has also facilitated the job of the end-user, not only by allowing the gathering, storage and processing of data, but also in communicating data throughout an organisation. It can also then be communicated to external databases beyond the organisation helped by the advent of technology such as integrated services digital network (ISDN) (Lee *et al* 1991).

Another major driving force (and one that is in some respects linked to the use of prototyping by full time system developers) is the end-user dissatisfaction with IS departments. They are often seen as too slow in delivering software regarded as key in vital decision making areas (Bray 1992, Welch *et al* 1992). Users understand far better than IS personnel the decisions they have to make, the decision-making process and the information needed to support those decisions (Rittenberg and Senn 1993). Consequently, they have, in many cases, undertaken direct responsibility for their own information needs.

An increasingly computer literate business population and the spread of new technologies (e.g. LANs, groupware products) means as Mclean and Kappelman (1992)

found, that PC managers are being stretched into areas once considered to be the sole province of Information Systems.

There is no question that UDA will continue to play an important role in corporate computing. However the risks that it is perceived to carry with it has led companies to question its value for projects of significant size and importance. Recent computing department re-organisations (Walker 1995) demonstrate that commercial companies are attempting to address the problems of performance and timeliness in undertaking projects, by radically changing computing account managers' job descriptions by making them directly answerable to end-users.

It is hoped that this new approach will lead to greater user satisfaction, and reduce the necessity for increasing UDA.

2.1.5 Rapid application development (RAD)

RAD can perhaps be regarded as a particular "brand" of prototyping. The DSDM manual (1995a) suggests that the first formalised form of RAD was practised by DuPont in the mid 1980's. It was described by them as RIPP (Rapid Iterative Production Prototyping), however a better recognition of the method came about with the publication of James Martin's 1991 book named Rapid Application Development (Martin 1991).

Since that point it has become a popular subject particularly in the commercial computing press (E.g. Beckett 1994, Baum 1992). The definition of RAD has much in common with certain definitions of prototyping. However the overriding principle behind the approach is to concentrate on meeting the true requirements of the business need when the system is installed (Martin 1991). This contrasts with the more traditional form of building software where the most important deliverable is producing a system that conforms accurately to a previously produced written requirement or specification.

Most sources of RAD material state that a typical project will take around three to six months to develop an operational system (E.g. DSDM 1995b). This time interval is perhaps the key factor in defining the RAD approach. It has been recognised that information systems which are delivered over a period of years rather than months will often fail to meet the requirements of the business. The increasingly fast moving nature of commerce means that often a business computing solution is required in order to

maintain or create a competitive advantage. Only by delivering systems quickly will this occur.

However speed of delivery is not the only characteristic that defines RAD. The DSDM (op cit.) consortium list a set of factors in their proprietary RAD method. The original version split these into thirteen principles, the second version reduced this to nine. Summarised these are as follows:

- Development teams are comprised of developers and users, who are empowered to make decisions
- Business requirements are more important than operational characteristics
- Developers and users keep in close communication through iterative development involving prototyping
- Change is encouraged and all changes are reversible
- All involved (developers and users) must be highly skilled and motivated towards business objectives - the focus is on delivery
- Testing is done throughout development and reviewed by both users and developers
- For larger projects frequent deliverables should be scheduled
- The high level scope and purpose of the system must be agreed and fixed early in development
- The relationship between vendor and purchaser must be one of co-operation

It can be seen from the above that the DSDM view of RAD comprises a high level development framework rather than a brand of systems development. It does not *prescribe* how computing solutions should be carried out, but *describes* the fundamental processes that are required in order to produce software which is useful for a given business.

As with prototyping a major part of the RAD process is to ensure that there is a high level of user involvement and responsibility throughout the project. This is another key part of the method. By involving users more throughout the development process it is essential that they are aware of the responsibilities that they have. There is evidence (Kinmond and Stephens 1995) that suggests that one of the main problem with projects involving prototyping is that the time spent involving both users and developers is sometimes considered to be used in a less than optimal manner. Hence it is important that users appreciate that their input into a project will have a significant affect on the likely success or failure of that project. Hence it may be the case that clients have as much responsibility for a systems failure or success as the development team. Unless the client organisation or sponsor understands this, a RAD approach will almost certainly not work.

2.2 Discussion

Although the previous section listed five separate forms of development, the reality is not so clear cut. Each form with perhaps the exception of UDA has an overlap with one or more other types of development. In particular more modern structured methods (e.g. SSADM version 4.2) indicate that there is not an insurmountable barrier between structured and RAD approaches to development (Hall 1995).

Nonetheless despite the cautious move towards more evolutionary and less prescriptive forms of development there is still reluctance in the commercial computing world to adopt these methods enthusiastically[1]. This is despite the evidence that methods involving prototyping and RAD do have advantages over more traditional methods and that business systems will be less than optimal if developed against a static and often incorrect specification. This point has been taken further by some authors, Lehman (1991) stresses that the

> "Effective software development process is essential for the economic and physical survival of society"

Further sections of this research work investigate prototyping and evolutionary development in more depth and consider how it can be used and controlled effectively.

[1]Holloway (1992) cites survey evidence about the popularity of various methods - especially SSADM in the UK.

Chapter 3. Software prototyping - An introduction

This section of the research considers software prototyping in more detail and looks at different definitions and approaches that researchers have used.

One area of confusion concerning the term "prototyping" is the very wide range of meanings that different authors have given it. Connell and Shafer (1989) point out that the term can even involve different procedures within the same organisation. An overview of some definitions used is presented in this section to illustrate this problem.

3.1 Definitions of prototyping

In order to examine the subject it is first necessary to examine some of the definitions that have been applied to the term. This Chapter starts with a consideration of the definitions that have been used by various authors as prototyping has developed as a subject of research and commercial interest. The section does not attempt to review every definition that has been put forward, but considers a sample which are considered to be significant.

3.1.1 Mason and Carey - The architecture-based methodology

The 1983 work of Mason and Carey does not define prototyping *per se*, although it does imply that prototypes will show the user a view of how the system will eventually appear and that prototypes themselves can form an important part of an application development methodology. Hence the suggestion is that prototypes (product) can be incorporated into the development of a system (process).

This idea is taken further with their suggestion of an "Architecture-Based Methodology". This is described in detail in the paper, however the components that define this methodology are:

- Architecture
- Transaction Screen perspective and dialogue-based design
- Project structure
- Iterative design

The components listed above have particular attributes which are described as follows:

Architecture

The authors consider that traditional methodologies are more concerned with the acceptability and function of the system rather than its understandability. Their methodology is primarily concerned with ensuring that the user has a clear view of how the developer has interpreted their requirement.

The designer works inwards from an external view to produce system details that are consistent with the outward appearance and the earliest stages of development should be concerned with finding a realistic expression of the system's appearance that is both *understandable and acceptable* (authors italics) to the users.

The external appearance must be contained in a form of specification that is capable of *complete and unambiguous interpretation* (authors italics) by the users, as well as providing the necessary details for the developers which will allow them to design and build the actual structure.

Unfortunately the authors do not provide details of how this can be accomplished, although they suggest that a screen oriented scenario which behaves like the proposed system can solve this type of problem for Interactive Information System (IIS) projects.

Transaction screen perspective and dialogue-based design

The authors suggest that a typical business system will consist of a series of "INPUT-PROCESS-OUTPUT SCREEN" sequences, and that this is key to developing IIS systems. It is suggested that a "scenario" of fixed screens is presented to the user in order to improve the communication between user and developer. It is also stated that this form of approach is not appropriate to other types of computing involving real-time, scientific processing or complex business processing.

Project structure

The methodology considers the management and control issues that are likely to arise in such a project. It states that three separate and distinct roles are supported, those of user, developer and architect. Each has separate responsibilities. It is interesting to note that the authors feel that the responsibility of the architect is to ensure that user(s)

understand the system specification and that the developers deliver the required product. They suggest that this role can be supported by software tools although the personal characteristics of the architect are important. It is stated that the architect must have the trust of all parties and is also responsible for the maintenance of the integrity of the development process by adhering to the methodology.

This methodology also considers the problem of who the architect should be affiliated to. It is interesting to note that the authors consider that in the "longer term" it is possible that the architect will operate as an independent agent, although they state that at the time of writing it is "clear" that the systems developer organisation should be the provider of this service.

Iterative design

The final part of the methodology is regarded as the essential part played by iterative design. It is argued that the users understanding of requirements will evolve rapidly as they are shown alternative approaches to applications during the requirements stage. The methodology proposed make use of three levels of iteration.

The first part involves the use of scenarios (see above). The user and architect iterate on scenarios until "an adequate first-level representation of the application is reached". The second level of iteration requires the use of database interaction and should be regarded more as a demonstration which assists more with the application logic and more detailed sequences of events. Finally iteration is required on the final specification of the system. this should include error-handling and recovery procedures. It will eventually form the basis for the final system specification.

Although this definition of a prototyping methodology is now over fourteen years old it makes an interesting comparison with contemporary thinking in this area.

The authors consider their version of prototyping to be a methodology. It has project management processes which are assigned to different parts of the project team. It also has a person or persons who are responsible for the overall progress of the project (the architect). In this respect it is similar to the most recent thinking behind RAD.

3.1.2 Floyd's definition

Floyd (1984) also made one of the first attempts to define and classify prototyping. It was noted (as have others since) that people have differing viewpoints on the subject. The extract overleaf illustrates this point, stating:

> "views differed as to the specific use of the terminology as well as the application-oriented emphasis on particular strategies"

However it was also commented that the different views seemed complimentary rather than contradictory despite the lack of a unique definition, hence:

- "A prototype is a simplified application system emphasising its interfaces - screens and reports - to the end user.

- Prototyping is a set of techniques supported by edp-tools for definition and verification of user requirements. Prototyping is an iterative approach, in which each version becomes better from the user's point of view"

It is then stated that prototyping can be further split into two types:

- Requirement - prototyping, where the prototype is made using a special prototyping environment which is eventually thrown away

- Implementation - prototyping, where the final version of the prototype is accepted by the user as the production version of the software.

It is then possible to further sub-divide implementation prototyping into software that will be used as the actual application, or that which is then regenerated in some way so that the code produced is more efficient and can then be used as the application system.

It can be seen that Floyd suggests that prototyping involves both an actual simplified system which can be shown to users and at the same time is a set of techniques that are used, hence it involves both *product* and *process*. In the view of the author this is a very important point. It illustrates that "a prototype" is a piece of software which can be either a very primitive representation of part of a system, or the full implementation, whilst "prototyping" is the collection of processes that constitute the activities that lead to the production of that software.

3.1.3 Hektmatpour and Ince - prototyping classification

Hekmatpour and Ince(1986) suggested definitions of rapid software prototyping based on how the prototypes would evolve and be used.

They proposed 3 types of prototype:

- Throw away prototypes are only constructed with evaluation in mind. This method, they suggest is most useful during the specification and design phases with very high level languages being used as the tool. The prototypes are not designed to be implemented as parts of the finished system. They are built quickly to demonstrate particular features.

- Evolutionary prototypes are built in a fashion that is really the complete antithesis to standard software development models. The software is introduced gradually and can be changed as the organisation itself changes. This change process affects the original requirements, so the system must be changed to cope. The development should be dynamic and iterative with errors picked up early on. The approach can be highly modular and will form the basis of a "proper" system. In order for it to be successful it is important that the software produced has modifiability built into it from the start.

- Incremental prototypes are based on a single design, with one section being implemented at a time. Functionality is gradually increased and the prototype can end up as the full system implementation. It differs from evolutionary prototyping because the requirements are fixed at the start and remain constant throughout the process. It is suggested that as a result incremental prototyping is easier to control and manage, but that it has less scope for adaptation compared to evolutionary methods. [1]

The categorisations of Hekmatpour and Ince define prototypes in terms of what they are used for. They state that it is very important that the development team should be absolutely clear about the nature of the prototype and its planned use. This should be decided prior to development and be understood by both the team and the customer. They note that the design and implementation of an evolutionary prototype is likely to

[1] Incremental prototyping is discussed in depth by Graham (1989 and 1992). A similar type of approach is also advocated by Brown, Carson, Montgomery and Zislis (1988).

be very different from a throw-away prototype and that problems will arise if this decision is not made.

A similar form of classification is defined by Graham (1991). Prototyping is considered to fall into three categories in some ways analogous to those used above. The categories are:

- Evolutionary prototyping is used where the target system may well be constructed in the same language as the initial prototype (with additional 3GL extensions if appropriate)

- Revolutionary prototyping is used where the prototype may end up as a throw-away item

- Revelationary prototyping is a subset of revolutionary and is research oriented for use with specialist tools in areas where proper budgets and cost controls are not paramount. It is more akin to hacking than a planned form of development.

Although Graham divides his definitions into three types it is suggested that the final category is artificial and that in reality there are only two major forms of prototyping worthy of consideration in commercial software development.

3.2 Two tier classifications

Within this sub-section it is perhaps appropriate to conclude with the summary definition given by Carey (1990).

It is suggested that prototyping can be categorised according to two approaches.

- Type 1 which is described as "iterative", and involves using the prototype as the final system after a series of evolutionary changes have been suggested by users input.

- Type 2 which is named throw-away and uses a model of the final system built in a 4GL as the basis for the implementation which is built in a 3GL.

Although the author implies that type 2 prototyping must involve a 4GL and does not necessarily involve iteration, these are not universally held views. However the use of

only two categories clearly illustrates the significant difference that the term prototyping has come to mean. There is work which expands on both of these basic definitions. For example Eaves (1991) considers that there are at least three forms of prototyping which fall into Carey's first category alone.

Using a similar categorisation Lichter, Schneider-Hufschmidt and Zullighoven (1993) suggest that prototyping has two fundamental characteristics:

- Prototyping is an approach based on an evolutionary view of software development, affecting the development process as a whole

- Prototyping involves producing early working versions ("prototypes") of the future application system and experimenting with them

The following section looks at examples of work which conform to the two models of prototyping outlined by Carey and considers what they seek to achieve within the broader field of software development. As in the previous section it is not possible to consider all work that has taken place, but examples of both approaches are examined.

3.3 Prototyping for information requirements

The United States Department of Defence Science Board Task Force on Military Software considered various problems of software development in 1987. As a result of this a Common Prototyping Working Group was set up to consider and lead a research program into the development of a prototyping facility. In 1988 a draft report on the requirements for a common prototyping system was published (Balzer 1988). Contained within this is a thorough definition of prototyping. It states that:

> "**Prototyping** is the process of constructing software for the purpose of obtaining information about the adequacy and appropriateness of the designers' conception of a software product. Prototyping is usually done as a precursor to writing a **production** system, and a prototype is distinguished from a production system by typically being more readily adapted, less efficient and/or complete, and more easily instrumented and monitored. Prototyping is useful to the extent that it enables information to be gained quickly and at low cost."

This definition is much more limited compared to some of those illustrated earlier. It suggests that prototyping is a process to do with learning about a problem and that it is not a process that leads to a direct implementation as incremental or evolutionary methods suggest is possible. Tanik and Yeh (1989) also advance this form of approach stating:

> "The final product of the prototyping activity is a working model that can be used for many purposes such as requirements validation, feasibility study of a complex system, behavioural specification of a system, and functional specification of a system design."

A similar example of this form of prototyping is the approach advocated in SSADM version 4 (Eva 1990). This proprietary structured method uses prototyping as a specification validation process as step 350 which is towards the end of the specification phase and assumes that most of the requirement is correct and in place. Prototyping is an activity which plays a comparatively minor part within a much larger structured method.

ADISSA (Shoval and Pliskin 1988, Pliskin and Shoval 1989) also uses a structured technique to develop the initial design stage which is then followed by a set of parallel prototyping steps (database, transactions, inputs/outputs, menus) allowing the users to react to the proposed specification. The relationship between prototyping, system specification and the software life-cycle remains as an area of research. Shoval and Pliskin (1988) suggest that prototyping in large systems is difficult because of the lack of structure compared to more traditional design methods. Their framework starts with structured analysis and design. This provides the basis for prototypes which are used to give the user an iterative process of understanding and at the same time refine and improve the analysis and design.

In further work (Pliskin and Shoval 1989) Structured Prototyping (SP) using ADISSA is advocated. This is a seven stage analysis and design method combined with a four stage prototyping method plus a further final stage where simulations with test data are studied. From this it would appear that control is imposed on the process by imposing a highly structured framework more akin to a structured design method. This is perhaps moving away from the iterative, non-sequential view of prototyping which is normally understood to be its major strength.

Another approach was advocated by Parks (1987). He stated that prototyping can be defined in terms of functionality. He gives three different levels:

- Input/Output design models. These have no working functions.
- Standard Functional Models. These have input and report screens and have some programming behind them which contain many features of the proposed system.
- Complete Functional Models. These have input and report screens, most of the system functions and real or simulated data is gathered and input into databases.

He states that the essence of prototyping is:

> "Providing a tangible example of the requirements of the final system which in turn can be used to specify the requirements of the final system".

This definition again shows that in the view of some practitioners prototyping is limited to a clarification and information gathering exercise at the start of the development life cycle and that the production of the software for the actual system is a separate activity. Purtilo, Larson and Clark (1991) extend this slightly by suggesting that:

> "Prototyping is an experimental category that typically precedes development of a software product, and is intended to reduce risk of failure for that product".

In some cases there appears to be a clear contrast between function and interface, however research carried out suggests that it is not always easy to define a prototyping exercise as either one or the other. Buckley *et al* (1988) state that the use of prototyping early on in the design process using the user interface, is needed to elicit requirements from users. They noted in researching the user interface for an estimating system that:

> "While a good user interface is desirable, the real value in using the user interface in this manner is to enable the end-users to talk about what they want the system to do, and to detect the problems before the main system is built".

The researchers felt that function and interface were not separate issues and that the two need not be considered in isolation. So it can be seen that the decision to concentrate on

prototyping the interface can also be taken in an attempt to discover information about the functionality of the system as well.

The examples above all confine prototyping to the "throw away" classification that Carey considered. Thus it is reasonable to conclude that although constrained in scope, this form of definition is valid for using the term prototyping for a comparatively limited domain of software development.

3.4 Evolutionary development methods

In contrast to the above examples there are also examples of work where detailed evolutionary development methods have been produced which build on and enhance examples of those that were produced in the 1980's. Three examples are discussed.

3.4.1 Boehms spiral model of development

Boehms spiral model (1986) describes the software process as a four stage iterative mechanism which is controlled by a risk driven approach to development, each cycle of the spiral comprises the following parts:

- The objectives of the portion being elaborated (i.e. performance, planned functionality)

- The identification of alternative methods of implementing the required section (i.e. different designs, re-use, customisation of purchased component)

- The consideration of the constraints that exist (cost, time, system architecture, programming language)

- The evaluation of the risks involved by the adoption of the different strategies.

Boehm states that the spiral model can accommodate any appropriate mixture of specification-oriented, prototype-oriented, simulation-oriented or other approaches to software development where the strategy is chosen according to the relative values of the risks involved and the effectiveness of the techniques in minimising those risks.

Boehm also states that an important part of the process is to review each completed phase of the cycle in order to ensure that all parties are committed and confident of the

approach to be taken to the next phase in the cycle. The model is illustrated in figure 3.1.

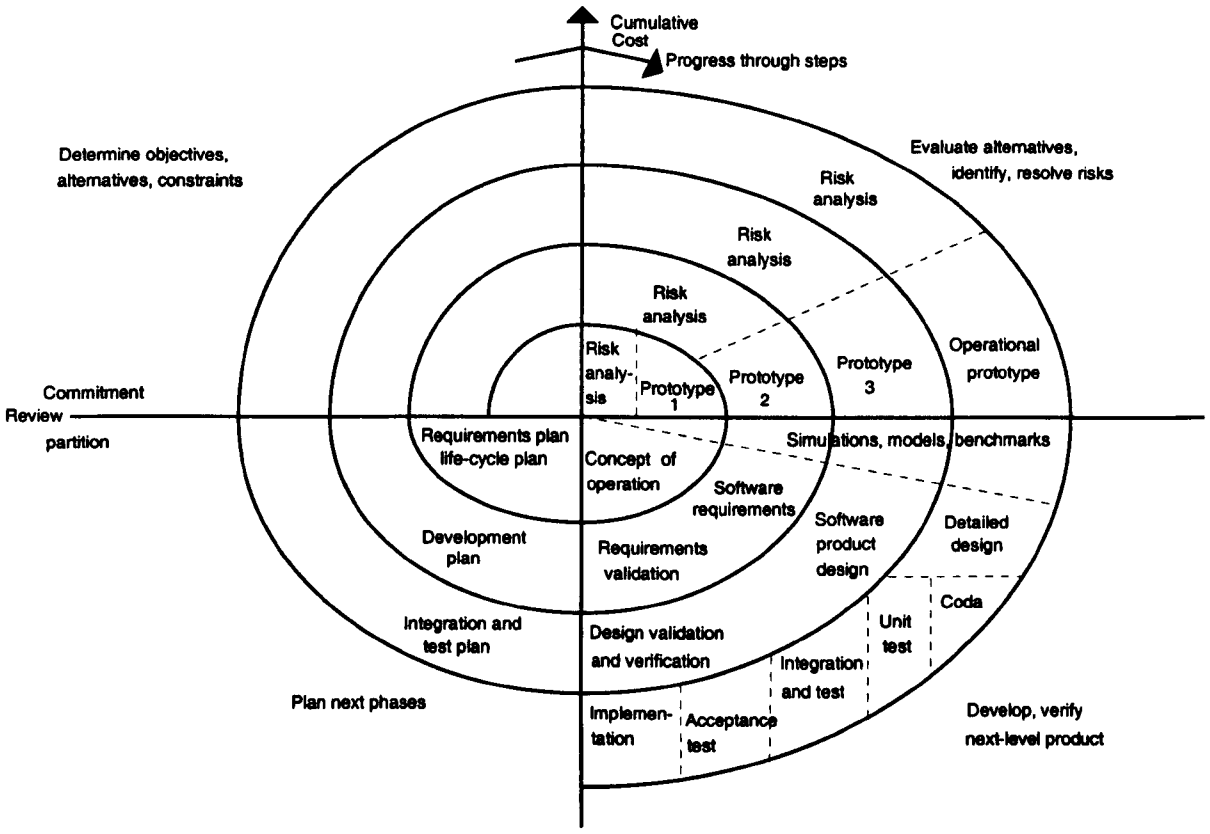


Figure 3.1. The spiral model of software development (Boehm 1988: © 1988 IEEE)

Boehm states that this type of development acknowledges that the building or evolution of a specification does not occur in a uniform or formal manner, that prototyping is involved as a risk reduction exercise at any stage of the process, and that it can accommodate "go-backs" to any earlier stages as better alternatives or new risk issues are identified.

Boehms model is very much a high level framework onto which a team can map appropriate processes that are suitable for the project/and or environment in which development is taking place. It is a good example of a non-prescriptive development method where iteration is acknowledged to be important and software difficult to produce without appropriate consideration of the different risks involved.

3.4.2 The DSDM life-Cycle

A more recent definition is provided by the DSDM manual (DSDM 1995a, DSDM1995b) which states that prototypes built as part of a RAD development will eventually become part of the final system, but that they will also be incremental in that they should be robust enough to evolve through a series of iterations. From this it is suggested that prototypes fall into four main categories which are:

- Business - relating to the business functions that require to be prototyped
- Usability - demonstrating user interface aspects without affect on functionality
- Performance and capacity - assessing the systems ability to handle the required amount of processing
- Capability/technique - examining a particular approach to the system design

It has been suggested that the large number of tools available that enable prototype systems to be produced can be used to create systems that prototype system function or the human- computer interface. The former concentrates on results from the system, the latter on the appearance of the system to the users. This type of classification has a similarity to that suggested by Parks (1987).

The DSDM consortium describe a default life-cycle which again conforms to the non-prescriptive iterative method as advocated by Boehm. It is based on five of the nine principles (see Chapter 1) which form the core of the DSDM method. These five are:

- The focus is on frequent delivery of products rather than on activities.
- Iterative and incremental development is important to developing systems rapidly
- Rapid application development must involve user involvement
- Testing is throughout the process
- All changes are reversible

DSDM state that apart from the iterative development, it is the very high level of user participation throughout the life-cycle that distinguishes the method from other approaches.

It is stated that prototyping within RAD is about communication, the use of Joint Application Development (JAD) workshops is about communication and that locating users in the development team is also about communication.

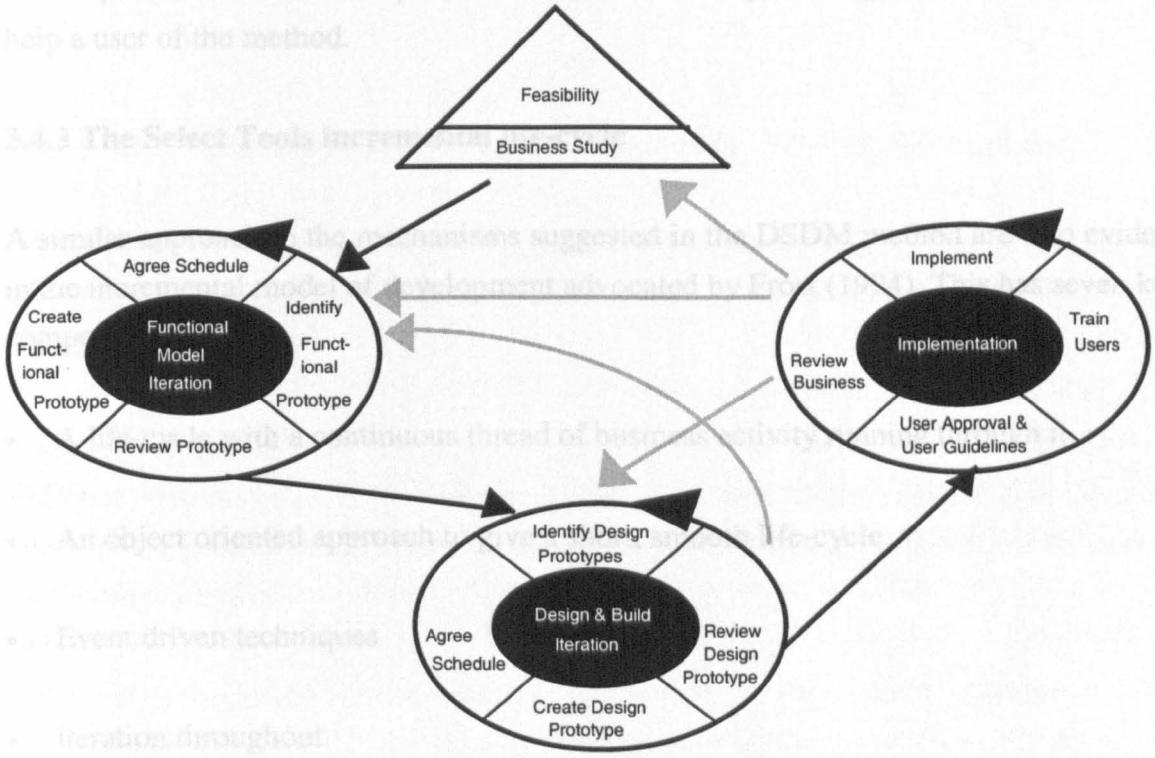


Figure 3.2. The DSDM development model (© DSDM Consortium 1995b)

The DSDM method has more inherent structure than Boehm's model. It starts from a feasibility stage and moves to a business study (the two of which can be combined). As with most traditional methods theses are primarily concerned with defining the highest levels of functionality required and examining the major business entities that are likely to be affected. Also as with Boehm's model, it is suggested that alternative approaches to development are examined.

After this has been carried out the main part of the life cycle takes place. The diagram illustrates that there are two iterative prototyping cycles. It also shows that this approach encompasses both requirements oriented prototyping designed to focus on eliciting the correct information required for the system and a second round of prototyping which is primarily concerned with ensuring production quality software is

produced. Although the diagram suggests that the divide between sections is clear cut the DSDM method mentions that in reality there will be some blurring between phases. It also suggests that each cycle can be navigated a number of times although a maximum of three is recommended.

As with Boehms model this method does not define the activities that are required to produce the various products. It is suggested that they will vary according to the nature of the product under development. However it does provide guidance which should help a user of the method.

3.4.3 The Select Tools incremental life-cycle

A similar approach to the mechanisms suggested in the DSDM method are also evident in the incremental model of development advocated by Frost (1994). This has seven key components:

- A life-cycle with a continuous thread of business activity running through it.
- An object oriented approach to give a short, smooth life-cycle
- Event driven techniques
- Iteration throughout
- Prototyping used throughout the method from the outset
- Incremental delivery supported
- Estimation can be carried out at several stages

As with the DSDM approach this method involves two distinct iteration loops within the life-cycle, one at the feasibility/business needs stage where there is a requirement for a close relationship between analysis and the prototype and secondly between the detailed design and the implementation and delivery of a particular increment.

The proponents of the method state that by ensuring a short life-cycle the problems of increment integration problems are minimised - although no details of how this are achieved are given.

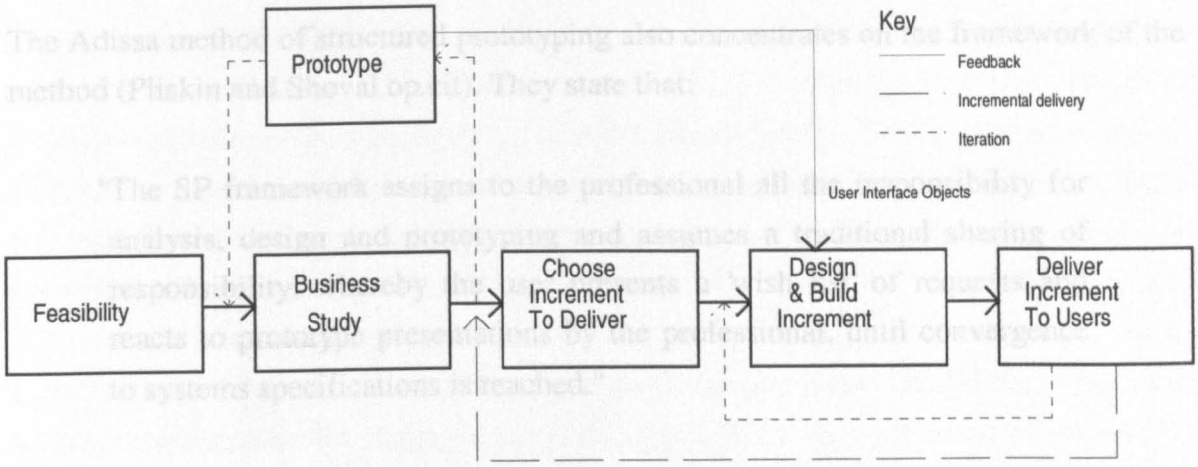


Figure 3.3. The Select Tools life cycle (Frost 1994: ©Select Software tools)

Unlike some other incremental or evolutionary techniques, this method advocates the use of a particular object modelling technique - OMT as described by Rumbaugh (1991).

Frost (1991) quotes Gilb (1988) by suggesting that developers should consider *how little can be spent in delivering maximum value or utility to users*, rather than the more traditional approach which asks *how much can be achieved given a critical constraint such as time or cost*.

When using the method in practice it is stated that the object model produced is not split up to provide the basis for incremental delivery, but that the division is better done by considering the users' requirements expressed directly in terms of stated business activities. By doing this it is felt that the increment delivery is prioritised on the basis of true need.

3.5 Planning and control issues in published prototyping literature

It is evident from examination of the definitions and models of prototyping above that the issues of planning and control are approached in different ways. This section will consider the issues briefly to give an overview of these different perspectives.

In the early work in the subject (e.g. Mason and Carey op.cit) management and control are not directly addressed, the emphasis of the work is very much on the communication issues between users and developers and the way that the methodology can also be supported by a particular tool.

The Adissa method of structured prototyping also concentrates on the framework of the method (Pliskin and Shoval op.cit). They state that:

> "The SP framework assigns to the professional all the responsibility for analysis, design and prototyping and assumes a traditional sharing of responsibility, whereby the user presents a 'wish list' of requests and reacts to prototype presentations by the professional, until convergence to systems specifications is reached."

However the mechanisms for ensuring that the project is managed and controlled in an effective fashion is not explicitly stated.

Boehm's spiral model (Boehm op.cit) emphasises that the entire software development project needs to be assessed against estimates of perceived risk. It is interesting to note that the author cites three areas where he feels that a "challenge" exists. It could be interpreted that one in particularly is related to issues of planning and control. He states:

> "In general, the spiral model process steps need further elaboration to ensure that all of the participants in a software development are operating in a consistent context."

He writes that checklists and guidelines are required to identify the most likely sources of project risk. Hence, by implication he is acknowledging that the planning of the project by using risk as the main controlling factor needs further elaboration and in that sense he seems to be very aware of the management and control issues that are so important.

Hekmatpour and Ince (op.cit) address the management issues directly and at the time of their work cite it as an important and ignored topic area. They discuss whether or not prototyping should be used in a particular project and state that the type of prototyping undertaken should be clearly understood. They indicate that adopting a throw-away or incremental approach should not prove to be too difficult in to software project management practises, but that the impact of an evolutionary approach is likely to be much more significant.

They consider the idea of dynamic contracting and also state that the need for discipline on a prototyping project is as great as on any other project. However they also state that at the time there was little data available that would allow a good comparison of these

issues. A number of suggestions connected with the way that development teams are organised are outlined and finally the issues of tools, user training, user time, channels for user feedback and user requests are also mentioned briefly. They do not recommend a particular model for development or even give details of techniques that have been successfully used, but they do recognise that management and control are important and under researched issues.

Two more contemporary development methods (Frost op.cit and DSDM op.cit) suggest different mechanisms for management and control. Frost as with some of the earlier work cited above concentrates on how the method (in terms of its activities) is constructed to ensure effective development of systems. Management and control are not featured as particular issues, although the use of Function Point Analysis (FPA) is recommended as a useful technique for estimating purposes, although it is noted by the author that:

> "we cannot expect a high level of accuracy!"

It is suggested that the estimates can be refined as the project progresses. Beyond the recommendation that FPA be used for initial estimation purposes, the method does not recommend particular project management techniques.

The DSDM approach to project management and control is strongly linked to the use of timeboxes[2]. It is stated that without the use of timeboxing the focus of a project team can be lost and hence project control will be compromised. The application of timeboxes within the method are outlined in the DSDM manual (DSDM op.cit). However as with most aspects of the DSDM method the description of their use is at a high level and examples are not actually cited. It is worth noting that at the time of writing the DSDM world wide web site (DSDM 1999) has a series of papers listed for publication in the future. The list includes two subjects (Project Planning and Reporting, Timebox approaches) that are part of the project management and control of the method. From this it is reasonable to conclude that more work in both areas is under way to enhance the DSDM process.

[2] Timeboxes are outlined in Chapter 9. A further discussion of the DSDM approach to management and control is included in Chapter 5.

3.6 Discussion

From the above it can be seen that prototyping encompasses activities which range from simple exercises concerned with clarifying user requirements at the front end of a conventional life cycle development, through to evolutionary development methodologies which entirely replace conventional staged methods. Although the definition of Hekmatpour and Ince involves an interim classification (incremental prototyping), this should be regarded as a sub-set of evolutionary prototyping.

Within the scope of this research it is therefore intended to conform to Carey's basic model but with certain extra clarification. Hence prototyping is defined as *the process of building software involving a high degree of user participation and feedback such that a product evolves in an iterative fashion.* Within this broad definition prototyping should be sub-divided into two distinct categories.

The first category is requirements prototyping. Here software is developed purely for the clarification of requirements or specification and *will not* be used as an implementation. It is not mandatory to throw away software produced during this process as it could form of a specification or vehicle for test generation, hence the term throw away prototyping is eschewed despite its relatively wide use in the literature. The examples of Balzer 1988, Parks 1987 and Eva 1990 all conform closely to this definition.

The second category is perhaps better termed as iterative evolutionary development. It is the process of building software that *will* be used in implementation and is constructed in such a manner that this is planned from the outset. The examples of Boehm (1986), DSDM (1995b) and Frost (1994) all conform to variations of this definition.

It can be seen that the second process can (and indeed does in some cases) include the first but that the converse, while in theory possible is not desirable.

Since the 1980s when prototyping first started to enter the literature it is arguable that the actual nature of prototyping has not changed significantly and that there are still the two basic definitions which have existed for over ten years. However it is apparent that more modern prototyping "methods" have attempted to elaborate on the definitions and in particular the category of iterative evolutionary development now has several

published "methodologies" which are designed to replace the structured methods that originated in the 1960s and 1970s.

The management and control of prototyping is considered in different ways by different researchers. It is another area where work is currently underway (e.g. DSDM) and hence it is not really possible to outline a recognised standard for prototyping management and control as so much variation exists in the published literature.

3.7 Prototyping and specialist tool environments

In addition to the process and product dominated work on prototyping, by the beginning of the 1990s it is apparent that there was also a considerable interest in the idea of prototyping linked to specialist tools and environments. Various research projects have been undertaken where a specialist tool has been used or developed and the results of its use documented in the literature. The section that follows looks at some examples of this.

3.7.1 CASE driven prototyping

Vonk (1990) noted that despite the plethora of published work in the field, the actual progress in the subject seemed slow. He defines prototyping as:

> "an approach for establishing a systems requirements definition which is characterised by a high degree of iteration, by a very high degree of user participation in the development process and by an extensive use of prototypes."

In this respect, his ideas fall in to line with those of Balzer and Parks discussed earlier. However after prototyping has been used in the requirements phase Vonk considers that prototyping is:

> "Essentially a technology-driven approach"

He suggests that prototyping is facilitated by the 4th generation languages and that in addition the use of CASE is also very important. He suggests that not only is CASE without prototyping ineffective, but that prototyping without CASE is almost impossible. He does not consider prototyping to be a methodology and in this respect does differ from some of the earlier definitions outlined in this Chapter.

3.7.2 TOPOS

TOPOS (Bischofberger and Pomberger 1992) is a toolset that is designed to support all the activities that occur within a prototyping-oriented life cycle. In order to use the method and toolset it is necessary to divide the cycle up into a requirements definition process, and a design and implementation process. Each part of the cycle is then supported by a different part of the toolset. The authors argue that there are two approaches to constructing prototyping tool sets.

- A single tool which provides all support for the development of all high-level and functional application parts OR

- To use various tools to develop different kinds of application part with a further tool to manage the co-operation of the other tools.

They consider that the second method has advantages in terms of generality and flexibility. They feel that different combinations of tools can then be used according to the application domain under investigation. Hence TOPOS conforms to this model. It comprises the following components.

- High level tools allowing requirements analysis and specification to be declared quickly, particularly for areas such as the user interface and data management.

- An exploratory programming environment which allows exploratory architecture validation and system implementation in different parts of a prototyping development.

- A project management environment which provides tools for the management of system documentation and other components which are produced during development and maintenance.

In addition the environment must be extensible so that other tools can be incorporated according to the application area.

It is argued that by using this form of environment that once the requirements definition process is terminated the software can be developed incrementally without disruption as parts need not be thrown away.

3.7.3 CAPS

CAPS (Computer aided prototyping systems) is a framework that has been proposed by Luqi from 1988 onwards (E.g. Luqi, Berzins and Yeh 1988, Luqi 1989, Luqi 1991, Luqi and Steigerwald 1992, Luqi and Cooke 1992). It is suggested that it should be used to:

> "...prototype large, parallel, distributed, real-time and knowledge based systems because the requirements for design of such software systems are difficult to assess"

It is stated that prototypes are built to gain information in order to guide analysis and design and support rapid generation of production code, hence conforming to Carey's first type of prototyping category. The CAPS is used to construct prototypes that the user evaluates against expected outcomes. If the prototype executes in an incorrect fashion, the user and designer identify problems that require correction.
The CAPS tool comprise the following components:

- An execution support system
- A rewrite system
- A syntax directed editor with graphics
- A software base
- A design database
- A design management system.

The tool can then be used to add changes to a prototype, to retrieve components from the software base, to generate production code if required, to assemble production systems, and to manage the prototyping process using the database. The tool designers state that groups of people working concurrently to evolve a software system requires sophisticated tool support to aid their task and that prototyping manually cannot be as effective.

In later work (Luqi 1992) it is suggested that by using this form of development that two areas of risk can be minimised in software development. Firstly prototyping can ensure that errors can be picked up early in the requirements process, thus lessening the chance of costly rework further on in the project. Secondly reviews of non axiomatic specifications (i.e. those likely to change over time, Lehman 1980) [3] can be undertaken in order to assess their current validity and that prototypes are very useful for this purpose.

The prototyping framework is constructed of five stages which make up the software process model. Initially the system is divided into two sections; axiomatic and non-axiomatic specifications. The specifications are validated with the users using prototypes. The final production system is produced from the specification that the final prototype illustrates and is then validated using the previously agreed specification. Finally the software is maintained in a proactive manner. I.e. All specifications are periodically reviewed to determine their current state of correctness and that any changes required are validated through the prototype which will identify the units of production program that require change.

3.7.4 Tools in General

Although there is extensive literature devoted to work in the development of specialist prototyping tools[4] it is not universally accepted that they are required for effective prototyping in software development. Smith (1991) states that he is not convinced that specialist tools are essential and that in some cases a tool choice which is too far outside the knowledge and experience of the development team may well lead to unnecessary overheads and higher risks in the project. It is also interesting to note that there is survey evidence that suggests that contrary to widely held views, standard 3GL languages are the most widely used prototyping tools (Martin and Carey 1991). This point is covered in more depth in Chapter 4.

[3]See also Lehman 1988 and Lehman 1991 for a more detailed discussion of uncertainty in specification.
[4]Baümer et al (1996) is a comprehensive recent example of this type of work.

3.8 Conclusion

By the end of the 1980s prototyping had become a well established topic in both the academic and computing trade press. However as Mayhew and Dearnley(1989) point out most of the work published concentrated on one of three areas:

- The advantages and problems associated with prototyping
- A discussion of a prototyping tool and its suitability in use
- The description of a project with an ad hoc use of prototyping within it

In particular they were concerned with the problems of introducing prototyping into commercial organisations and felt that the lack of practical advice on how to adopt prototyping was one of the reasons why relatively few organisations had adopted the method for the development of systems.

If it is accepted that Mayhew and Dearnley are correct in their observation then it perhaps necessary to consider what the major advantages and disadvantages of the technique are and also to examine its uptake in commerce. This is examined in the next Chapter. Only by looking at prototyping in use (or the lack of) is it possible to suggest methods by which it can be both improved and advanced.

Chapter 4. Benefits, pitfalls and commercial use of prototyping

Chapter 4 considers the evidence presented in the literature with regard to the claimed advantages and disadvantages of using software prototyping. This is carried out by reviewing not only published case study and experimental evidence, but also by reviewing a series of surveys carried out in the subject area (Section 4.4).

Brooks in his landmark paper of 1987 identifies that building software will always be a difficult process and that the most difficult part of it will be determining precisely what should be built. He states:

> "No other part of the conceptual work is as difficult as establishing the detailed technical requirements, including all the interfaces to people, to machines and to other software systems."

Unfortunately evidence from contemporary computing journals is a constant reminder that this is still the case and that unfortunately development teams still run into serious problems when designing and implementing systems.

A 1990 survey (KPMG 1990) illustrates the magnitude of the problem.

- 62% of all companies surveyed had projects that ran significantly over cost or time
- This figure rose to 71% in companies of over 500 employees
- Technical issues were the main problem in only 7% of the cases cited
- 85% were problems considered to be as a result of poor project management, or management issues such as a failure to define system objectives, a lack of communication or a lack of familiarity with the project scope.

It would appear that any methods that attempt to rectify these problems would be welcomed by professional software developers. Prototyping is often cited as an approach that addresses these issues. Hence the next section considers the perceived advantages and disadvantages of using prototyping and examines why it is perhaps not used in software development to the extent that might be expected.

4.1 Why use prototyping?

In most cases the proponents of prototyping cite a range of advantages, typically these would include examples comprising the following or closely related factors:

- It should enable good communication between the user and developer,
- Requirement changes can be made at an early stage,
- Gradual introduction with increased user acceptance should be possible,
- Development time should be reduced, system testing, debugging, and modification should be less as the user is kept close to the project at all time.

However as the quantity of literature which addresses the advantages and disadvantages of prototyping is considerable it is worth examining the published work in more detail and then evaluating the survey evidence which has also been carried out.

4.2 Benefits of prototyping

Examples of the perceived benefits of prototyping documented in the literature include the following topics, each is briefly examined in the following section:

- To allow more effective communication
- To aid in risk reduction
- To increase the knowledge of users and developers
- To facilitate easier training and increase the confidence of users
- To increase the effectiveness of testing
- It is a more natural form of software development
- To increase the speed of development
- To reduce maintenance costs

4.2.1 An effective communication tool

There is a considerable quantity of published material that describes how communication can be improved by the use of prototyping (See Morrison 1988, Mayhew 1990, Stephens and Bates 1990). Examples of how particular researchers have described this are illustrated below.

Brittan (1980) gives a summary of the use of prototyping in development:

> "The final system may well have features of one or more of the prototypes, but the basic philosophy is to learn, to find out, to discover."

This theme is also stated by Guitierrez (1989) who considers that any experimental form of development should always be considered as a learning vehicle for providing more precise ideas about objectives and targets.

Thomas (1987) gives a good summary of the use of prototyping in expert systems development by stating:

> "The value of the prototype should be that it communicates to the users and to senior management the likely scope and value of the final system, that it communicates quickly to the expert how successful the knowledge base so far constructed could be and that it enables an accurate assessment of the scale of the project to be made."

Harris and Parker (1987) give evidence that also supports this view. They note that a system (LIVES) did not live up to expectation because no prototypes had been designed and there had been virtually no consultation between the developers and users as the design had progressed. A similar stance is taken by Martin (1991) who contends that the normal method of depicting system function and behaviour will not be as effective as demonstrating an actual working piece of software.

Software engineering has developed formal methods[1] which attempt to specify software mathematically thus ensuring that the implementation is "valid" (i.e. is a correct instantiation of the specification). Although this is without doubt of great importance it is not practical for users (unless highly trained in the notation used) to verify that the

[1]Vienneau (1997) reviews the state of formal methods after two decades of development.

specification is indeed what they require. In these cases prototyping can be used as a method to demonstrate that the software conforms to the expectations of the user.

The theme of communication is also examined in surveys which have been carried out. These are examined in more detail in a later section in this Chapter.

4.2.2 Risk reduction

Several authors consider that prototyping is an important technique when used to try and reduce the risks associated with software development. This has already been mentioned briefly in Chapter 3 (Luqi 1992). Other authors including Harker 1988, Graham 1989, Bates 1989 and Sommerville 1992 all consider the positive affect that prototyping can have on the identification of areas of potential risk within a proposed system.

By allowing users an input into the decision making process and allowing them to evaluate alternative strategies, better informed decisions can be taken by project managers. This can then help evaluate ways of minimising the risks involved.

4.2.3 Increased knowledge of users and developers

By using prototyping users and developers can learn more about the problem domain as prototypes can be criticised (Hilal and Soltan 1991). Harker (1988) states that prototypes are useful in that they allow users the opportunity to respond to alternative design decisions at a stage where they can still be influenced and amended. Budde and Zullighoven (1990) also encourage this theme stating:

> "Systems development is a learning process for all parties involved, and not a transformation of a specification into a target system."

Mayhew 1990 suggests that not only does prototyping allow learning about crucial elements of a system from the outset, it also allows participants to increase the width of their knowledge as the process goes forward.

The lack of a place for learning is illustrated well by Jorgensen (1984) who states:

> "Systems development is also a learning process. One of the most striking features of the traditional approach to systems development is its

failure to include the learning that takes place during the development process"

The ability of users to state what they want in a more effective manner is described by both Ince (1988) and Bates (1989). This ability is likely to have come about because of their direct contact with a prototype which has allowed them to learn more about the process that they are expected to participate in. The developer is more likely to encourage useful feedback by demonstrating prototypes in a realistic fashion that allows a good understanding of what is required by the user community. However it is also the purpose of the prototype to educate the software engineer as well (Harwood 1989).

4.2.4 Easier training and increased confidence of users

As users have participated in the design and development of the system it is argued by some (Hekmatpour and Ince 1986) that training costs will be lessened by taking a prototyping approach and that users will be more amenable to the system being implemented as they will already have been involved with it during development and already have a sense of "ownership" of the system. This and similar points are made by many authors including Dearnley and Mayhew (1983), Mayhew (1987), Bates (1989), and Smith (1991).

In addition if an incremental approach is taken then an early payback is possible as parts of the system can be delivered earlier than if a monolithic approach was used. This is illustrated in both Graham (1989) and Stephens and Bates (1990)

4.2.5 More effective testing

Testing can be spread out more evenly over a prototyping project. Hilal and Soltan (1991) suggest that verification of a product is more effective using a prototype compared to a written specification. Certainly some aspects of testing should be more effective using prototyping, although in the case of throw away prototyping it will not be easy to simulate system performance or related issues.

4.2.6 A natural method of software development

Arthur (1992) contends that rapid prototyping as part of a rapid evolutionary development method is the only way that software can be created for successful systems over short time spans. He states that the construction paradigm will not succeed because

it will always take longer and that the information necessary to build it is not available at the outset. As a result only by using evolutionary methods can developers learn about the system in an effective manner. A similar point is made by both Bates (1989) and Smith (1991) where they contend that undergraduates favour this form of approach when building software.

4.2.7 Increased speed of development

Using the results of a limited experiment involving seven graduate development teams Boehm, Gray and Seewaldt (1984) suggest that systems can be developed faster, saving as much as 45 percent of the development effort compared to systems built using a structured approach. In addition the products were smaller (usually around forty percent of the size of comparative structured systems). Boehm also shows that developers using prototyping spent less time implementing unnecessary functions ("gold plating").

Unfortunately, this is one of only very few examples where assertions about productivity are backed up by empirical evidence. Although other studies (e.g. Goodwin 1993) claim increased productivity the examples of projects cited cannot be easily compared against those carried out using comparable methods.

Hekmatpour and Ince (1986) state that speed and cost of prototype construction are of great importance although this is in the context of producing a prototype that is used for experimentation and learning rather than for extended use.

4.2.8 Reduction of maintenance costs

Vonk (1990) considers that maintenance costs can be reduced particularly where there is a high level of uncertainty regarding the application area. He suggests that to embark on a prototyping approach where there are low levels of uncertainty is probably not cost effective. Unfortunately there is a lack of published work in the area of prototyping and maintenance so most evidence in this important area is of a speculative nature - a point emphasised by Smith (1991).

Boehm *et al* (1984) contends that the systems constructed using prototyping in their study were "more" maintainable than those produced using specifying, but as pointed out earlier this study was only of a comparatively limited scale.

Connell and Shafer (1989) also make claims about the likely decrease in maintenance costs using their structured prototyping method. Unfortunately there is little evidence to back these assertions and hence their empirical validity is questionable.

4.3 Prototyping pitfalls

With the long list of advantages often given, it might seem curious that the technique is not more widely used. The question of why this is the case has not been widely addressed in the literature on prototyping and hence a definite answer is not available.

Typical difficulties are listed below and then briefly outlined in the next section.

- Problems of introduction and use
- Undue user expectation
- Friction between participants
- Time and project responsibility of clients
- Performance issues
- Inadequate analysis of the problem domain
- Implementation of throw away prototypes
- Estimation and resource planning
- Contractual difficulties

4.3.1 Problems of introduction and use

Although prototyping is widely written about in both academic journals and the computing trade press it is only very recently that a standard commercial methodology for its use within a RAD framework has emerged (DSDM 1995a, DSDM 1995b). Mayhew *et al* (1989) concluded that most published work in the area tends to fall into 3 distinct categories:

- The advantages and problems associated with the use of prototyping.
- A particular tool and its suitability for prototyping.
- A description of ad hoc adoption of prototyping for a particular project."

They stated that little attention has been given to the problems of introducing and organising prototyping into a typical commercial environment. Overall they suggest that the lack of a formal methodology is one of the main reasons for its slow adoption by commercial management. If this is the case then hesitancy within commercial environments is hardly surprising. Tozer (1987) typifies the concern that commercial developers tend to show by stating :

> "For corporate systems, evolutionary development can mean never quite knowing what the final system will do, when it will be ready or what it will cost."

This type of problem is also discussed by Vonk (1990) and is also illustrated by Smith (1991).

Carey (1990) suggests some guidelines that can be used by developers for deciding which applications are more suitable for prototyping; however there are many questions that can be raised about the possible problems using prototyping and the lack of a formalised method:

- Prototyping is a highly iterative process, at what point should the process start?
- As the process encourages iteration at what point should the iterations stop?
- What documentation is required to develop a specification if the prototypes themselves are representing part of the final system function?
- Is there a final specification if the process is dynamic?
- It is very difficult to discover instances of prototyping projects that have failed, should this be interpreted as a sign of infallibility?

Unfortunately many of these issues have not been addressed in detail and it is possible that the use of prototyping has not spread due to the uncertainty surrounding its use.

Hayball and Barlow (1988) in a research project on expert system development methods acknowledge the fear of developers regarding rapid prototyping stating:

> "...in the past, KBS (Knowledge Based Systems) of any complexity have largely been developed in research environments using ad-hoc techniques of rapid prototyping which engender the likelihood of unconstrained iteration."

Guimaraes (1987) discusses similar issues and states that:

> "Prototyping proponents who thought the method would do away with documentation, project management, and the systems development cycle itself are beginning to realise that just the opposite is true."

Prototyping has been used successfully in various projects. However there are still issues which require more research in order that a disciplined approach can be adhered to. The adoption of methods by software developers wary of the current lack of structure are likely to be more enthusiastically received if the methods can be shown to be controllable.

4.3.2 Undue user expectation

This problem can occur due to a variety of different phenomena. Carey (1990) considers that delivery dates can be too optimistic, the final system can have inefficiencies that have been glossed over in the design phase, the interfaces may have been rushed and the full depth of the problem may not have been analysed. Other authors (e.g. Mayhew 1987, Livesay 1984) state similar problems. There is also survey evidence (E.g. Alavi 1984) that illustrates this point. Further survey work concerned with this issue is discussed later in this Chapter.

In all case it is important that the development team make clear to users and clients what the likely risks and problems are going to be. This is most important in order to promote a more co-operative relationship between the different parties. If the users are to form a more active part of the decision making process they are required to be kept well informed and must not be given to expect more than that can be delivered.

An example of this type of problem is if an early prototype is built using a sophisticated GUI when the final implementation is likely to be built on a more traditional machine. Often interfaces associated with relational databases have not been as flexible as other environments and users should not be misled if the final implementation is to use such an environment.

4.3.3 Friction between participants

As the team involved in the process of development is likely to include a significant user element the scope for friction between the different parties is likely to increase. In some respects this can well lead on from undue expectations having been created (see above), but as Mayhew (1987) points out the prototype can itself be misused in order for the development team to try and wilfully manipulate the users involved. This problem is also discussed by Smith (1991) and by DSDM (1995b) both of whom highlight the possibility that users who feel threatened by the process can be difficult to work and communicate with, which can lead to a lack of understanding between parties and even conflict.

4.3.4 Time and project responsibility of clients

The time needed by prototyping will involve not only developers, but users as well. Unless client organisations are prepared to allow users to fully participate in the prototyping process their valuable input will be lost. The increased user participation required also means that a failure to accept this responsibility will lead to project difficulties that can no longer be placed entirely at the door of development teams. It is important that the time used in prototyping sessions is used in a sensible and responsible manner by both users and developers.

Problems of:

- Developers dwelling on unnecessary trivia needlessly repeating similar sections of the system or

- Users not being prepared to communicate (whether wilful or due to intimidation)

will lead to significant problems. It is important that mechanisms are in place that can prevent this happening. Grottola (1989) suggest that clients can prepare by allowing sufficient time in their work schedules, ensuring that the most appropriate users are

involved, not prevaricating when important issues are at stake, and being critical of all components. Williams (1992) also cites evidence of waning user enthusiasm as a project progresses and states that developers must maintain good progress to sustain user enthusiasm.

4.3.5 Performance issues

The problems of system performance after using prototyping is discussed by Prizant (1986). However the main theme of the argument is that fourth generation languages and relational technology whilst particularly suitable for prototyping can incur performance overheads. Although other authors also hint at the same thing it is important to appreciate that prototyping is not inextricably bound up with these types of software and that there is evidence (see next section) that prototyping is mainly carried out using third generation technology. In the view of the author performance is more likely to be affected by inadequate design, poor choice of software environment or under specified hardware. It is possible one of these could be influenced by the choice of prototyping as a design method, but prototyping itself will not lead to sub-standard system efficiency. Carey (1990) contrasts a successful prototyping project with one that had severe performance problems. He concludes that the success of the second project was due to significant advances in tool technology and that as software tools geared for prototyping become more efficient, the problems of performance inefficiencies will decrease.

4.3.6 Inadequate analysis of problem domain

The temptation of development teams to produce working software to demonstrate to users can well lead to the analysis phase being rushed. After this has happened there is then likely to be further pressure to implement software which has not been adequately documented or tested. This problem is particularly true where throw away prototypes appear to users to be adequate for the required task. Stephens and Bates (1990) elaborate on the pressures that can arise due to this problem.

It is important that the correct type of user are selected by the clients in order to help allow the developers to learn about the system. Users who are unwilling to co-operate will create difficulties (see above), but there are published examples (Braunstein, Lauer and Doane 1991) where middle management are used to represent typical users of a system leading to:

> ....an almost cavalier attitude towards the treatment of end-user information requirements within a system."

Another problem can arise when too much emphasis is placed on implementing an inefficient system in order to fulfil particular user requests. Although prototyping must encourage the fullest participation with users, sometimes it is necessary for development teams to show that alternative strategies are more effective. This problem is illustrated in Rzevski (1984).

Hilal and Soltan (1991) argue that a prototyping approach can lead to a narrower and shallower knowledge domain. It is stated that the range of options is reduced earlier as fast feedback limits possible solutions, hence a global view is sometimes lost.

4.3.7 Implementation of throw away prototype

The pressure on developers to deliver a working system based on software that was originally only intended as a throw-away prototype can lead to problems. This is mentioned in Stephens and Bates (1990). If developers do not resist this pressure the delivered system is likely to be unreliable, inefficient and will probably be difficult to maintain. Scott Gordon and Bieman (1994) state that managers must maintain a firm commitment to the chosen prototyping paradigm and fully define the prototype's purpose and scope at the outset to avoid this problem.

4.3.8 Estimation and resource planning

It can be argued that the iterative nature of prototyping with out the apparent easily identifiable milestones prevalent in structured techniques make estimation and planning particularly difficult. Boehm and Pappachio (1988) argue that this is not necessarily the case and that appropriate guidelines can be formulated. This particular issue is discussed in more depth in Chapter 9.

4.3.9 Contractual difficulties

Software for projects is normally delivered as part of a contract that stipulates the deadlines involved and the fees payable on delivery. It has been recognised that building systems to a fixed price can often only be achieved if no change occurs within the project. (Pers comm Martyn Thomas 1990[2]). If prototyping is used and

[2]Martin Thomas in the keynote address at the 1990 BCS Conference on Software Engineering

change is to be encouraged in order to ensure that the software is fit for the anticipated task then this form of contractual arrangement is not always suitable. Graham (1989) illustrates this as a particular difficulty when projects involve one or more external software houses. Mathiassen (1984 In Budde at all) proposed a *dynamic contracting* strategy where contracts are initialised formalised but negotiated by interested parties during the development process. By revising the contracts as new information comes to light management can itself become more dynamic.

4.4 Commercial use of prototyping - survey evidence

It can be seen from the previous section that prototyping appears to have a lengthy list of both advantages and disadvantages. The next section considers issues of prototyping based on survey evidence from both British and North American sources.

There have been several surveys into the use of prototyping within the United States (Alavi (1984), Necco, Tsai and Gordon (1989), Palvia and Nosek (1990), Martin and Carey (1991), Hardgrave, Doke and Swanson (1993), Scott Gordon and Bieman (1994) Hardgrave and Wilson (1994)[3] and Hardgrave (1995)).

However to the author's knowledge comparable studies have not been carried out in the public domain within the United Kingdom. In order to discover more about the extent to which prototyping is used within industry in the United Kingdom a survey of 200 companies was carried out which was supervised by the author.

The results of this survey undertaken in 1994 by Kinmond which are analysed here, cannot be considered to be exhaustive, but it is hoped that trends can be noted and hence an indication of the way that software development is moving can be gained. The survey attempts to look at some of the major issues in prototyping, the languages used, the perceived advantages and disadvantages, and the subsequent use to which prototypes are put. Where applicable the results are compared and contrasted with findings from other survey work.

The early draft of the survey questionnaire attempted to contain the questions to one side of A4 and enable the respondents to merely tick appropriate boxes in response to a question. This had the advantage of being a familiar format, but the design appeared too cramped and did not give room for expansion on individual questions. The redesigned questionnaire was drafted to allow the respondents room adjacent to the

[3]The same data is also used as a basis for Hardgrave (1996).

question for a more free form response. Despite piloting the questionnaire, some sections were poorly received and did not appear to be well understood. Pre addressed and free post envelopes were included and a response rate of 20 % was achieved.

Table 4.1. Response to questionnaire

Study	Year mailed	Number	Number returned	% Response
Alavi	1984	Not stated	12	N/A
Necco, Tsai and Gordon	1989	Not stated	97	N/A
Palvia and Nosek	1990	300	65	22
Martin and Carey	1991	1000	71	7
Hardgrave *et al*	1993	500	96	19.2
Hardgrave and Wilson	1994	500	88	17.6
Kinmond and Stephens	1994	200	40	20

The 20% response rate for the survey is comparable with other studies.

The results for each question asked are discussed and illustrated in the following section.

Table 4.2. Primary activity of the organisation

	Primary activity	Number of respondents	% Response
1.	Agriculture, forestry and fishing	1	3
2.	Energy & water supply	3	8
3.	Mining, chemicals, metals & minerals	1	3
4.	Metal goods engineering, electronics & vehicles	6	14
5.	Computer manufacture	3	8
6.	Systems OEM	0	0
7.	Other manufacturing industries, Construction	4	10
8.	Retail, distribution, hotels, catering & repairs	1	3
9.	Computer dealer/reseller	0	0
10.	Transport & communication	0	0
11.	Banking, financial, business services & insurance	5	13
12.	Computer services bureau	2	5
13.	Software house	3	8
14.	Education, Health, government & local authorities	9	22
15.	Other	1	3

Table 4.2 shows the breakdown of the responses to the questionnaire based on the primary activity of the organisation . It can be seen that there was a good spread of different types of organisation. Public sector organisations were well represented, but

there was also a good response from manufacturing and finance. One organisation did not answer this question and is listed as category 15

Table 4.3. Job function of the respondent

	Job Function	Number of respondents	% Response
1	Technical Customer Support	4	10
2	Systems Programming	2	5
3	Manager Of It/ Mis/ DP Department	6	15
4	Systems Analysis Or Design	4	10
5	Programming(Development/Application)	5	12.5
6	Systems Analysis And Programming	7	17.5
7	Overall Head Of It /DP	0	0
8	Networking	0	0
9	Operations	0	0
10	Software Engineering	6	15
11	Hardware Engineering	2	5
12	Board Level Director For It/Mis/DP	0	0
13	Database Administration	1	2.5
14	Other	3	7.5

This section was included in order to assess the value of the respondents replies. The survey results demonstrated that over 50 % of respondents worked in the area of systems analysis, design or programming (categories 4,5,6, and 10), the second largest category of respondents was managers of IT/MIS/DP departments (category 3). The authors feel that the survey has therefore been answered by staff that are in a good position to reflect on the questions asked.

Table 4.4. Software development method used by the organisation

Development Method		Number of respondents	% Response
1.	SSADM	9	24
2.	YOURDON	6	13
3.	JSD	3	8
4.	Multiview	0	0
5.	Information Engineering	1	3
6.	None	21	52

The respondents were asked to state what development methods were in use by the organisation. The results to this question raised a variety of points:

- A large percentage (52%) of respondents stated that they were not following a methodology.

- Comments concerning methodologies revealed that many popular methodologies are only partly used or greatly modified. (For example several organisations stated they used SSADM version 4 and were using prototyping in an evolutionary manner. This approach is not in line with SSADM (V4) methods).

- Several organisations claim to be following more than one methodology.

- There did not appear to be any particular relationship between use of a method and size of the organisation.

As mentioned above, comments written on the questionnaire ranged from:

"The methodologies employed may be altered in their use according to the developers preferences, but the basic guidelines are implemented."

Through to:

"Several methodologies are used, there are no strict guidelines."

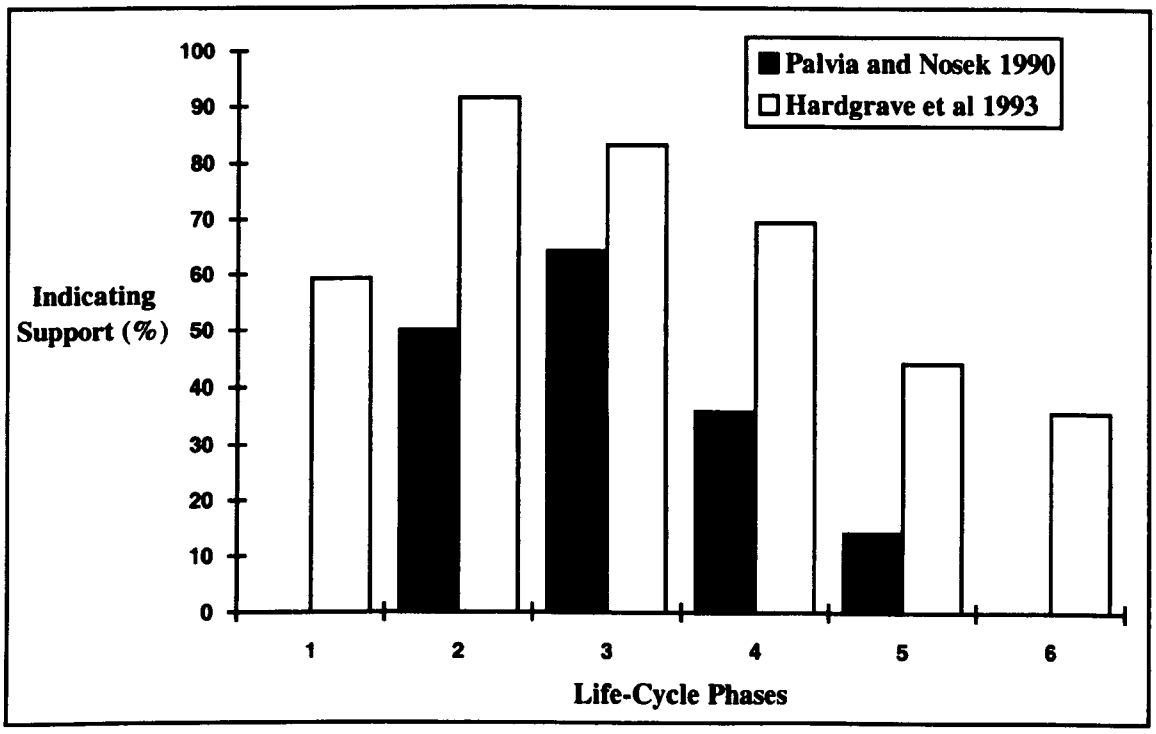
The results and comments above suggests that the use of recognisable development methods still have some way to go before they can be considered to be normal practice within software projects of this type. This study is broadly in line with one that is reported in Computer Weekly (Green-Armytage 1995) where SSADM accounted for 16% of the number using a methodology whilst 33% used their own in house methods.

Three of the studies (Necco *et al* (1989), Palvia and Nosek (1990), Hardgrave *et al* (1993) investigate the use of prototyping within a structured approach. Although the research methods are not directly comparable the following conclusions can be drawn from the work carried out.

Prototyping need not replace a structured approach, but can be used within it. Necco *et al* (1989) discovered that this was the case for about half of the organisations they surveyed.

Both Palvia and Nosek (1990) and Hardgrave *et al* (1993) investigated the use of prototyping within particular phases of systems development. The former considering it as a four stage process (analysis, design, development, implementation), the latter a six stage process (feasibility included at the start and post-implementation review at the end). The graph below indicates the findings.

Graph 4.1. Prototyping use within life-cycle



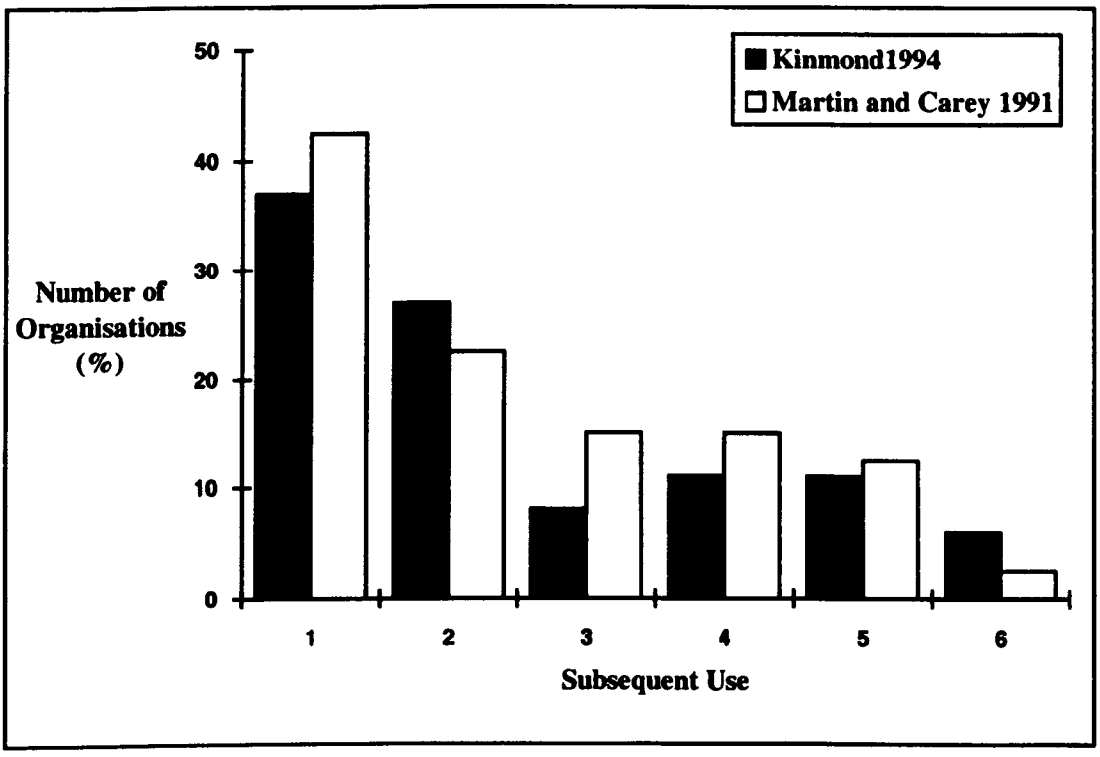
1.	Feasibility Study
2.	Analysis
3.	Design
4.	Development
5.	Implementation
6.	Review

Key to prototyping use within life-cycle

From Graph 4.1 it can be seen that the general trend exhibited by the two data sets is that the use of prototyping is more prevalent in the analysis and design phases, than later stages. The Hardgrave study indicates much higher use of prototyping throughout the cycle. This is perhaps because the study was carried out more recently, although other (unknown) factors could be influencing the results shown. The Palvia and Nosek study indicates a peak of prototyping use in the design, rather than the development phase - again it is possible that the use of prototyping has increased substantially in the period within the two studies and that increased prototyping in analysis require less design verification. The very high usage of prototyping in the analysis phase in

the Hardgrave study leads the authors to suggest that the prototype and associated documentation could become the requirements document.

Graph 4.2. Subsequent use of prototype

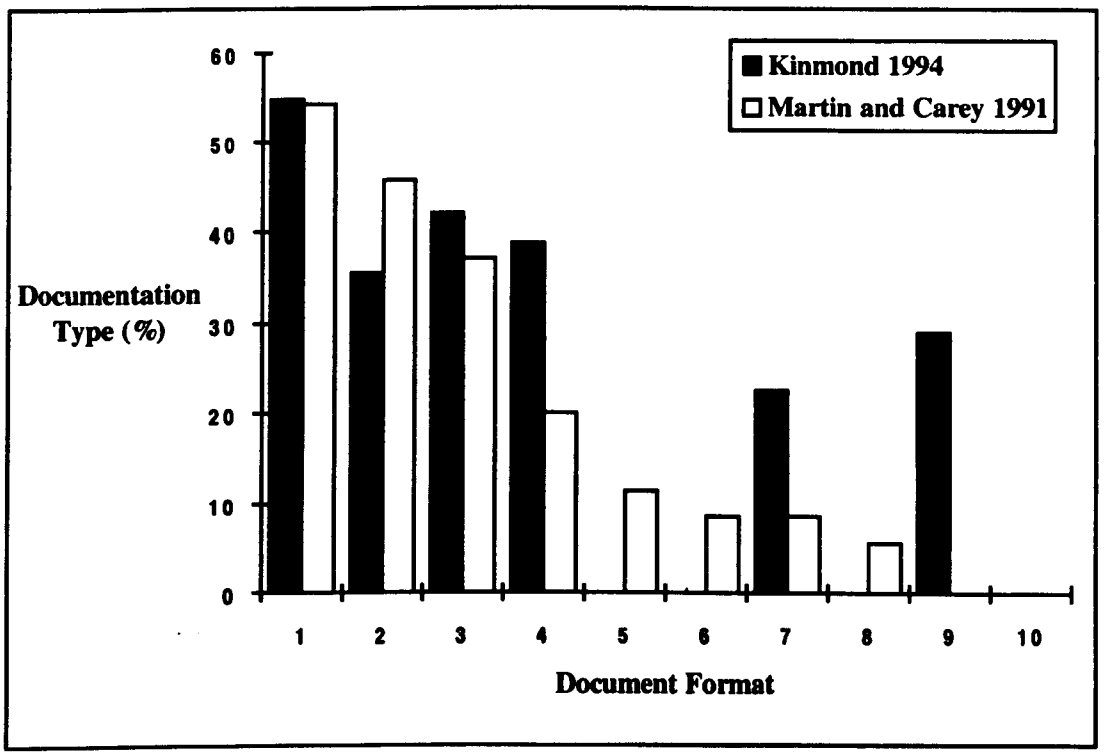


1. Becomes operational system (evolutionary)
2. Is used for demonstration purposes
3. Code is re-used
4. Is thrown away
5. Is used for training purposes
6. Is used as part of the system Documentation

Key to subsequent use of prototype

It was possible to give more than one response to this question. In the graph the two sets of data are plotted as percentages to give clear comparison. The more recent (UK) survey shows slightly fewer systems evolving into the final system, but more of the prototypes have code in them that is re-used. Overall a high proportion of the organisations put the prototypes to some use and only 11 % throw them away.

Graph 4.3. Documentation of prototypes



Number on Graph	Documentation type	Option in Martin & Carey	Option in Kinmond
1	Data flow diagrams	Y	Y
2	Data dictionary	Y	Y
3	System Flowchart	Y	Y
4	Program flowchart	Y	Y
5	Text (narrative)	Y	N
6	Hierarchy charts	Y	N
7	No documentation	Y	Y
8	Other types	Y	Y
9	Entity analysis	N	Y
10	Entity life histories	N	Y

Key to documentation type

This question attempts to find out how prototype models were documented. It was felt that alternative answers to this question should match the examples given by Martin and Carey. Pilot studies revealed that some adaptation was required for the United Kingdom sample population. More than one response was possible to this question.

The following points were noted:

- Data flow diagrams were most popular in both surveys.
- Data dictionaries and flowcharting techniques were widely used.
- Entity analysis was a popular choice in the UK survey.
- A significant number of respondents stated that no prototype documentation was produced. This represented 20% of the UK respondents.
- Entity life histories were not used by any UK organisation. This seems to conflict with the stated use of SSADM by several organisation.

Although documentation is carried out, it would appear that developers have very different views on what is the most effective way of documenting a prototyped system. The degree to which techniques for documentation are adopted as part of the prototyping process varies as the comments received show. These ranged from statements such as:

> "Documentation tends to be done after the product is complete and released to the customer."

Through to more worrying admissions including:

> "Prototypes are not documented because documentation is a lengthy process."

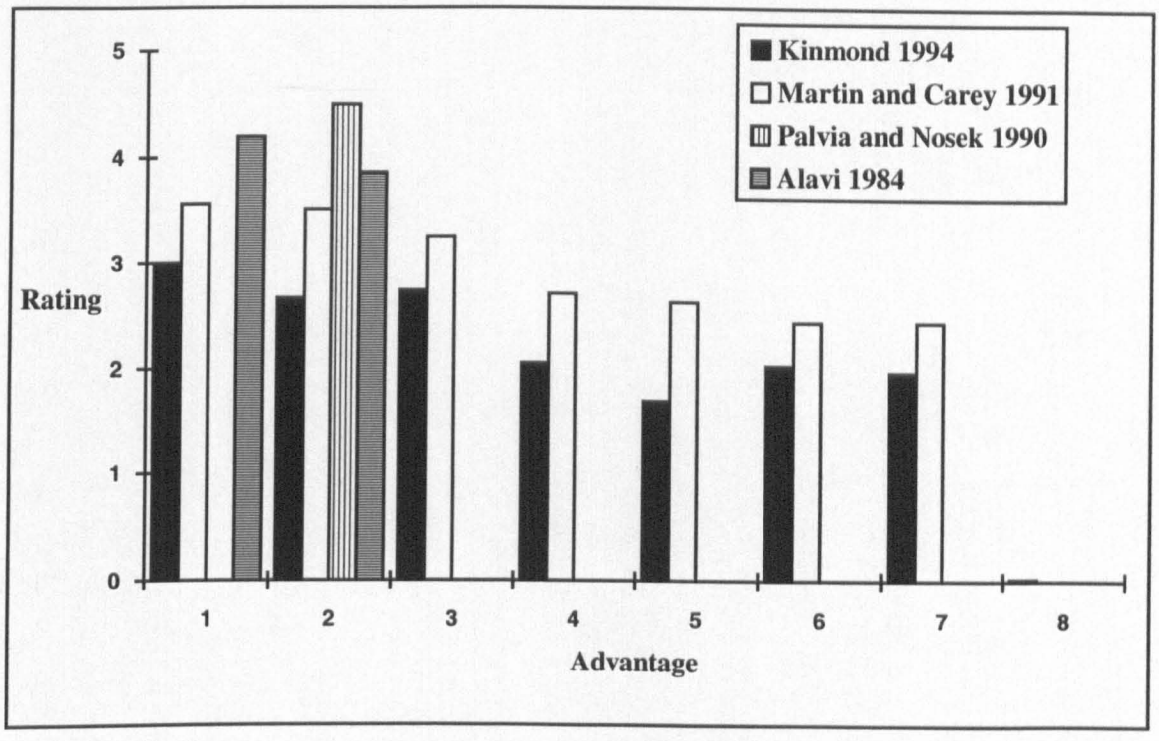
Yet the importance of documentation is also acknowledged in comments such as:

> "Software is developed on a variety of platforms, many different languages are used throughout the site. Seldom does more than one person work on the same application."

This final statement was made in conjunction with comments regarding the crucial part that documentation plays in the development of any quality system. It is possible that the adoption of more evolutionary forms of development require a re-think of the

way that documentation is best carried out. This is an area which is worthy of further study.

Graph 4.4. Advantages of prototyping



Advantage
1. User involvement
2. User communication
3. User satisfaction
4. Shorter development time
5. Training
6. Reduced development costs
7. Reduced Maintenance cost
8. Other

Key to advantages of prototyping

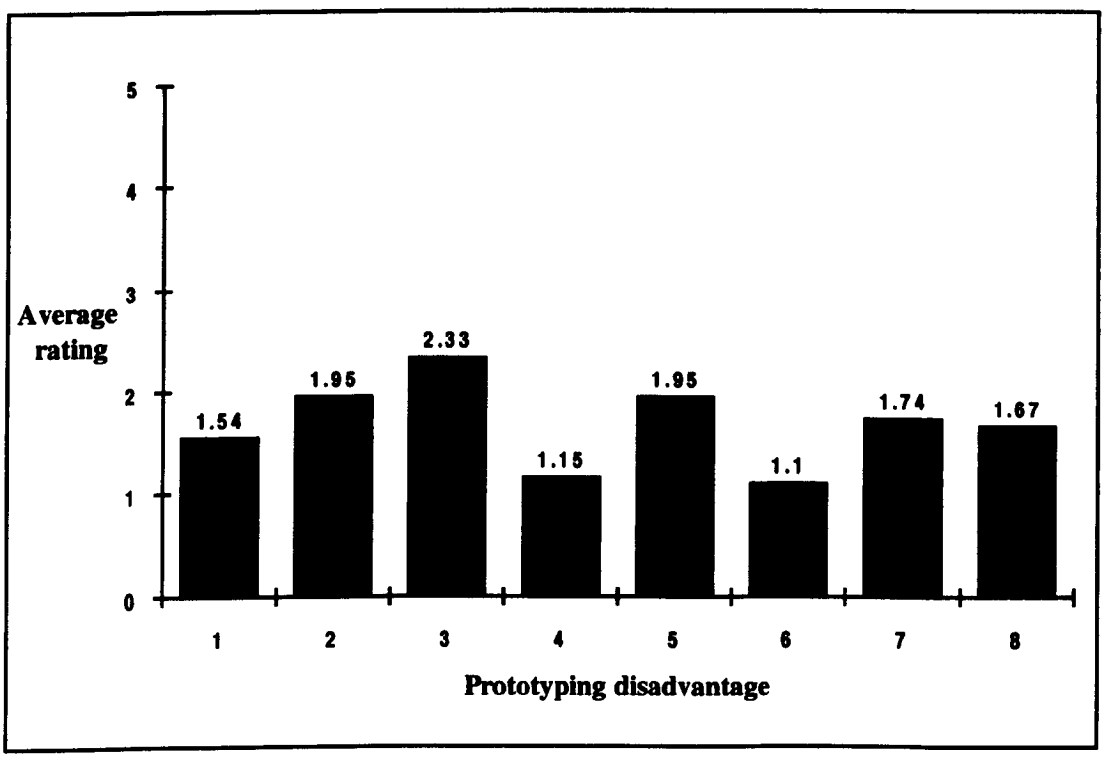
Respondents were asked to rank the advantages that prototyping gave them. A Likert scale of one to five was used. The graph is scaled so that a value of one represents a very unimportant advantage, whilst a value of five represents a very important advantage.

The graph plots the mean figures from both the Martin and Carey (1991) and Kinmond (1994). Although the absolute figures for the two surveys do not map on to each other, the trend of each set of figures is very similar. This suggests that both sets of respondents have similar expectations and experiences on the use of prototyping. In both cases the user centred nature of the process comes across strongly as the most

valued characteristic of prototyping. In both cases the reduction of both development and maintenance costs are not considered to be particular advantages of using prototyping. Figures from two other surveys are also included in those categories where the results are equivalent. The importance of user involvement and communication is marked as very significant.

The Palvia and Nosek survey also shows that communication between developers, the early discovery of requirements are both rated as very positive features of projects using prototyping. In guidelines suggested by Hardgrave and Wilson (1994) they conclude that prototyping is not used where the developer has wide experience of the application domain, but that it is used where there is a need for experimentation and learning prior to full commitment to a project.

Graph 4.5. Disadvantages of prototyping



Number on graph	Disadvantage
1.	Reduced management control of the project
2.	False user expectation
3.	Time required for user Participation
4.	Lack of necessary tools
5.	Increase in development costs
6.	Integration with existing methodologies
7.	Inadequate analysis carried out
8.	Poor documentation of systems

Key prototyping disadvantages

This question was asked to ascertain which characteristics were considered to be the most disadvantageous in using prototyping. Martin and Carey (1991) choose not to consider this points in their 1991 study. In this study a Likert scale was again used to rank the disadvantages of prototyping The scale used was again from one (very unimportant disadvantage) to five (a very important disadvantage).

The major disadvantage was seen as the time required for user participation. The results do not make it clear whether it is the user that feels his time is being wasted by developers being unable to focus on their requirements or if it is the developer who feels that time with the user is used inefficiently.

It is also noted that creating false user expectations and the possibility of increased development costs are disadvantages associated with prototyping, but with a low rating (1.95) they are not considered major disadvantages.

Integration within existing methodologies is not seen as a primary disadvantage, possibly because most of the respondents (52%) stated they were not following a standard development methodology. A similar level of disadvantage was also associated with specialist tool usage. This suggests that prototyping is not dependent on particular tools being available.

Necco *et al* (1989) discovered that the lack of standard prototyping procedures followed by users wanting to use the prototype as the production system were the two most frequently cited problems when using a prototyping approach. It is possible that their second reason is related to the creation of false user expectations as discovered in this survey. However the two surveys are not directly comparable, hence accurate conclusions should not be drawn from this. Alavi (1984) cited four perceived problems:

- Prototypes can be oversold creating undue user expectation

- Planning and control can be difficult. This was stated to be partly due to the "newness" of the prototyping approach which required different procedures compared to traditional methods.

- Better guidelines are required particularly for large systems. Boundaries and sub-division of systems need careful thought.

- There are problems of maintaining user enthusiasm, particularly after the highest priority requirements were satisfied.

Table 4.5 Languages used for prototyping and operational systems

Study	3GL alone	4GL alone	4GL converted to 3GL
Martin and Carey (1991)	22(55%)	6(15%)	12(30%)
Kinmond (1994)	24(61%)	11(28%)	N/A

This question was asked in an attempt to discover if there were any identifiable trends in the use of particular languages and their association with prototyping. In particular it was hoped to discover if the frequent association of prototyping with the use of 4GLs was in evidence in the survey.

The results revealed that several development languages (both 4GLs and 3GLs) were in use by the same organisation for both the prototype and the operational system, this prevented simple categorisation. Martin and Carey (1991) found that 55% of respondents used COBOL as the final implementation language and it was also the most common prototyping language. Kinmond (1994) found that in excess of 60 % used a 3GL as the final implementation language and that C was the most common language, followed by COBOL.

Both studies suggest that prototyping specialist tools or 4GLs are not seen as a prerequisite for projects using prototyping. This conflicts with evidence from Hardgrave and Wilson (1994) who concluded that evolutionary prototyping is not used unless prototyping support tools are available.

4.5 Summary

Although the results of the Kinmond survey have been compiled from a small sample there are still some trends which are worthy of comment.

- The survey appears to be consistent with many of the finding of Martin and Carey (1991).

- The majority of the organisations replying were not following a methodology and a high proportion of those that were following a methodology such as SSADM were modifying it.

- The subsequent use to which prototypes are put reveals that in the majority of cases the prototype was not thrown away, either evolving into the final system or used in some other way.

- Prototypes were documented in a wide variety of different forms but over 20% of respondents stated that they do not document their prototypes, whilst those that do, mainly use data flow diagrams. The popularity of entity analysis as a documentation technique may be indicative of the increased use of relational database technology.

- The advantages of prototyping centred around the better levels of user communication, user involvement and user satisfaction, this strongly agrees with the finding of Martin and Carey (1991). Training and reduced development time and costs were thought to be less important advantages.

- The major disadvantage to prototyping concerned the time required for user participation. Less significant disadvantages included the creation of false user expectations and increased development costs. There is evidence that pressure to implement throw-away prototypes is a problem (Necco *et al* 1989) and there are still question marks over management and control issues, however the latter does not seem to be solely a problem within a prototyping environment (Palvia and Nosek 1990).

- The results of this survey suggest that prototyping is used by commercial developers despite the concerns that are sometimes expressed. The author feels that many of the comments made by Martin and Carey in 1991 remain pertinent today.

- Hardgrave and Wilson (1994) suggest that evolutionary prototyping is not used unless appropriate tools are available. However both the Kinmond 1994 study and the earlier work of Martin and Carey reveals that specialist tools or environments are not a prerequisite for prototyping, these results suggest that Smith (1991) may be closer to the reality of prototyping in suggesting that special purpose prototyping tools are not essential.

The surveys examined did not focus on the relationship between prototyping and CASE. It has been stated (Vonk 1990) that it is impossible to prototype without having access to appropriate CASE tools. Unfortunately there does appear to be

empirical evidence that either supports or conflicts with this assertion. Further work requires to be carried out to investigate this claim.

4.6 Discussion

From both the general literature and the survey evidence there are some issues which come across strongly.

4.6.1 Advantages

Prototyping will strongly aid learning and communication, both for users and developers. There is also strong evidence that user satisfaction levels will be higher.

4.6.2 Disadvantages

User time required is undoubtedly increased if prototyping is used. The responsibility of users within a project is increased. If there are difficulties in ensuring sufficient commitment to a project from users, then there are likely to be severe project difficulties. Developers can be put under pressure to implement solutions which were never designed to be permanent - these pressures must be avoided. Documentation of systems using prototyping also seems to be an area of weakness, this could be connected with the view that unlike conventional methods which are perhaps over reliant on documentation for both modelling and project control, the entire relationship between prototyping and documentation is unclear.

Estimation, management and control procedures appear to be confused areas within the prototyping area. The problems of iteration and the mapping of conventional management techniques onto evolutionary methods seems to cause unease in commercial software development. There seems to be little evidence that supports the view of Boehm (1984) that software produced using prototyping leads to improved maintainability.

The following Chapter investigates the issues on management and control in more depth and examines the techniques that have been suggested by researchers and practitioners to allow more effective control over prototyping projects.

4.7 Comments on the use of survey work.

It is worth recording that there does appear to be a good degree of consistency between the various surveys that have been carried out. As has been recorded in this Chapter there are several features (both positive and negative) of the prototyping process that seem to figure consistently in most of the surveys examined. This suggests that surveys can be a useful research tool in the area. It is the opinion of the author that future surveys should try to explore the issue of documentation of prototypes, and in addition to try and find out more detail about the main disadvantage cited, i.e. how to gain maximum benefit from the time spent between user and developer. It might also be possible to try and gain further detail about some of the other problems that exist.

Unfortunately when carrying out this type of research, the amount of detail that can be derived from a questionnaire will always be limited, as increasing the scope of the survey will inevitably lead to a reduce in the response rate. Hence it is inevitable that surveys will always have certain shortcomings that affect their overall usefulness.

Chapter 5. Prototyping: management and control

The management and control of prototyping are examined in this section of the research in order to discover what theories currently exist to aid developers. A comparison of management and control mechanisms used in different forms of development are examined.

There is a view that although prototyping can be useful, it cannot be effectively controlled and that its use is particularly limited by management problems (see previous Chapter). Hence adopting prototyping as a development method in its own right is considered by some to be impractical. There is in contrast, considerable evidence that prototyping and evolutionary methods do have benefits over more traditional methods in other areas and that a wider use of them would benefit software developers in general.

5.1 Background

Research so far carried out, both in the author's experience (Stephens and Bates 1990, Stephens and Bates 1992) and in other work (e.g. Purtilo *et al* 1991, Morrison 1987, Mayhew 1991, Tate 1990) suggests that prototyping does have real benefits for helping to define requirement boundaries and to facilitate user involvement in and commitment to systems and system development. The way in which this is achieved is by a genuine sharing of responsibilities, between users and developers, for learning about system requirements and the control of the project. Other benefits are also often mentioned, these have been discussed in the previous Chapter.

Prototyping may be viewed as an investment in user participation with a large potential payback in system quality and development efficiency. There is, however, a widespread view (perhaps misplaced) that project management, particularly project control, is inherently more difficult when using prototyping. The following statements from Warhurst and Flynn (1990) represent this view which is often considered to be one of the fundamental drawbacks of using prototyping as part of a practical evolutionary development method:

> "An important criticism of prototyping is that projects using this approach are difficult to control and manage.[..] Whilst prototyping may be appropriate, it has however been conspicuous for its lack of control

> and documentation, two fundamental requirements for quality software production."

It is certainly the experience of the author that evolutionary prototyping projects present challenges in the approach taken to management and control. Upon further reflection it is not surprising that control becomes a major concern in a process that encourages iteration and where some control is inevitably handed over to the user. The author would agree with others that the approach required in evolutionary development is revolutionary compared to a more traditional life cycle approach (Ince 1988, Hekmatpour and Ince 1986). The approach taken by the DSDM consortium (DSDM 1995b) is also very much concerned with the management framework of the development process.

The evidence that prototyping is an uncontrollable and undocumented development process is not universal throughout the literature. As the previous Chapter illustrates survey evidence (Palvia and Nosek 1990) shows that structured approaches were only rated marginally higher than prototyping in the contribution made to project control. Additionally the ratings for documentation on both methodologies was not significantly different. This is also echoed by Martin and Carey (1991).

This Chapter considers the current control procedures which are recommended by practitioners of structured methods and contrasts these with practices which are advocated by prototyping research. Structured methods are chosen as the comparative development paradigm since they currently represent "good" software engineering practice, and experiences of organisations using them are available in the literature.

The later part of the Chapter will examine frameworks for the control of prototyping.

5.2 Project control

Within structured methods, project control is often considered to be a valuable side effect brought about by the prescriptive nature of the method. The underlying assumption is that as a project is divided into small steps and well defined phases, the control of the project automatically becomes manageable as a result. Hence prototyping, because it does not have well defined heterogeneous steps is seen to be difficult to manage and control.

Only by examining and comparing the control strategies involved using both forms of system development can a more objective judgement be made of their relative strengths and weaknesses. In order to carry this out it is necessary to investigate the mechanisms in more detail.

To carry out a comparison it is very important to differentiate between what a system development methodology is and how it differs from a project plan. Wells (1987) also suggests that this is an important distinction and is significant for project control.

Ince (1991) describes the main constituents of a system development methodology as a management framework which determines the control, monitoring and ordering of tasks. This is combined with a series of notations which specify the main parts of a system, i.e. the data, the processes, the system architecture and the user interface.

A project plan involves a schedule, which can be controlled and monitored by assessing work done in terms of deliverables completed, and the progress can (or should !) be compared against an initial estimate of cost or resource required. From this it can be seen that the two are not synonymous and that using a particular methodology still requires a plan in order to be effective.

This represents a very important point. It appears that implicit in the criticism of prototyping is the idea that because the development process is iterative rather than sequential, project plans cannot be constructed; and hence effective management becomes impossible. The converse is that the sequential nature of structured methods does allow planning and hence control can be maintained over the project. However most definitions of prototyping suggest that it can deliver either partially working systems, or a full working sub-set of a larger system as part of an incremental or evolutionary development. Additionally prototyping has been shown to be able to cope with large projects (e.g. Hekmatpour 1987).

It should not be more difficult measuring the progress of a project being carried out in an evolutionary way than with a structured method which sets out to build an entire system in a sequential but monolithic manner. This point also illustrates that it is very important to differentiate between the terms evolutionary development and prototyping. This point has already been discussed at length in Chapter 2.

Linkman and Walker (1991) advocate measurement using metrics as an effective way of controlling development. This type of approach would seem to be well suited to

monitoring and controlling prototyping. By adopting this strategy it might be possible to understand and control the trade-off that exists between an accurate and complete specification being available at the outset, and the evolution of those requirements with experience. Weber (1986) states that this is still a poorly understood area.

It is important to examine how prototyping is carried out in order to consider the validity of the opening criticism. By examining different control strategies it might be possible to suggest methods by which prototyping can perhaps have its unmanageable tag shaken off.

A project using any formal approach to management will start with a survey of the projected project. It will attempt to identify the main system boundaries, the project goals and the management and planning that will take place after initiation. Some investigation of the existing system might also occur, partly to help identify new requirements. These should be produced with due regard to the overall business (or appropriate equivalent) objectives of the organisation.

This research assumes that a basic review of the problem will occur irrespective of the method involved. Hence the examples illustrated below are from stages occurring after the initial survey has taken place.

5.2.1 An overview of control using a traditional structured method

The following section considers in some detail how typical structured methods suggest dealing with project control once the software development process has been initiated.

It is necessary to examine how a method can help guide a developer to make the decision to move from one part of the process to the next. The point at which this occurs represents a fundamental part of project control. Effective control and management should mean that this occurs at the "optimum point" in the project progression. Beyond this point development effort outweighs potential benefit.

Examples of how two well known methods tackle these problems will be examined briefly. Firstly Yourdon, secondly Structured Systems And Analysis Design Method (SSADM).

Yourdon

After the initial investigation, the next technical part of using Yourdon involves drawing up a series of data flow diagrams (DFDs). These attempt to represent the processes that occur within a system. The processes are linked by arrows which depict the flow of information between those processes.

Figure 5.1 is taken from a published case study (Yourdon 1989) and shows the top level DFD. It is representative of the level of complexity contained in such a diagram. In this particular case diagrams representing lower process levels are often qualified by additional notes.

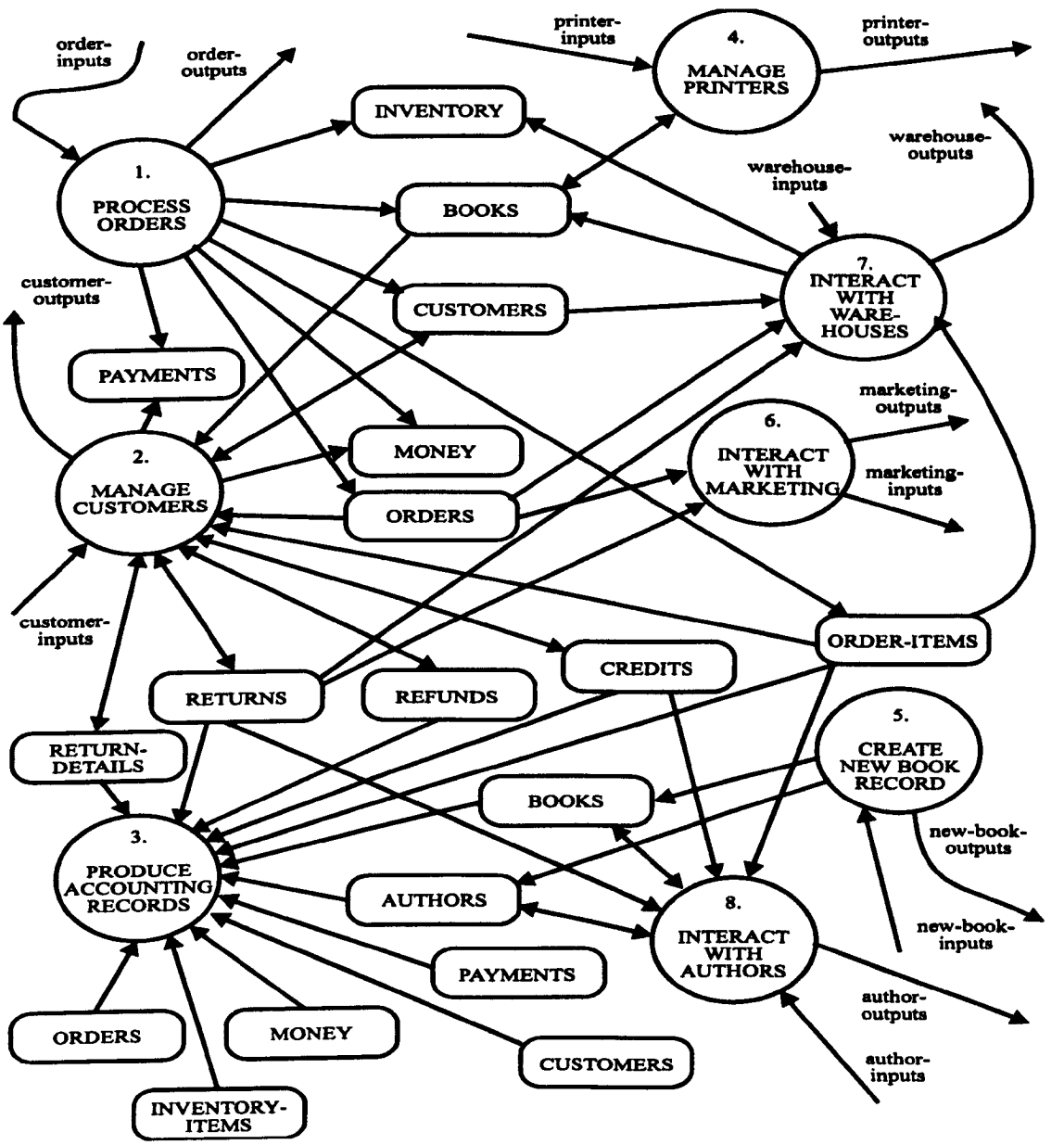


Figure 5.1. A representative top-level DFD. (From Yourdon 1989)

DFDs are constructed by obtaining information about the proposed system from the client (either from interviews, or from a written specification provided by the client). It attempts to model the system in a manner that the client can understand and comment on. The diagram is therefore an essential client/developer communication tool. In order for the developer to proceed further with the system, the client must understand not only the notation used but also the information portrayed in the data flow diagram. I.e. the client is required to go through a two stage process in order to produce an accurate mental model of the data flow diagram. As with any cognitive process leading to a mental model, the diagram will represent a different representation to different people. Johnson-Laird (1983) states

> "Obviously, a satisfactory blueprint for one individual may be grossly inadequate for another, since any set of instructions demands the knowledge and ability to understand them."

This difference of perceptions is well known to practitioners of both prototyping and Soft Systems Methodology and is discussed at length by Checkland and Scholes (1990).

Eventually a developer will be able to present a client with a comprehensive set of DFDs that accurately show the processes that model the system. If the client understands the notation, he or she will be able to point out any inaccuracies that are present in the diagram, the developer can then amend these. This process is repeated until the developer is sure that the diagram is correct.

Unfortunately the accuracy and control of the process will depend on the developer and client being able to verify that the diagram is indeed correct. The diagram is likely to end up incorrect if the client does not understand it or misinterprets any of the information that is within it. A similar problem will occur if several people are required to validate the same diagram. It might in fact appear to be "correct" from one clients viewpoint, but wrong from the perspective of another. Different people will have different opinions and priorities about how the system should appear. Hence the project management techniques used must be able to resolve these problems in order that the development processes can proceed.

Controlling this lies with the developers. It is their judgement that determines at what point the DFD construction should terminate and the next stage of the method should

commence. The use of the method does not take this decision away from the developers, nor suggest if the optimum progression point has been reached.

From the above it can be seen that the DFDs produced are more likely to accurately reflect the mental model that the developer has of the system, rather than those of the client or users involved.

SSADM

SSADM is a structured approach to systems analysis and design which relies heavily on the use of three fundamental views of a system namely DFDs, Logical Data Structures (Entity relationship diagrams) and Entity Life Histories. It consists of a core of five main activities: feasibility, requirements analysis, requirements specification, logical system specification and physical design. Analysis and design are each divided into three stages. Each major activity is divided into stages, steps and tasks.

Within it the basic function of DFDs is to show how data flows around a system. Downs *et al* (1988) state that the objectives of a data flow diagram are as follows.:

- To graphically document the boundaries of a system.
- To show the movement of data between the system and its environment.
- To provide a hierarchical functional breakdown of the system.
- To document the intra-system information flows.
- To aid communication.

As with the Yourdon method, the progress and control of a project will depend on a client understanding and validating a diagram representing the processes occurring in a system. The accuracy of the requirements specification and the progress of the project will depend on the ability of the user to interpret and comment on the diagrammatic representations produced by the developers.

From this it can be seen that SSADM demands a similar level of understanding of the notations used compared to Yourdon. Hence the accuracy of the model and the control methods are also determined by the same processes.

The problem with SSADM in that the user participation process is designed primarily to be an attempt to produce a system that exhibits completeness and consistency (Hogarth and Biggam 1994). SSADM is used to produce a system that is "correct" rather than one that is perhaps the most appropriate.

Yourdon (1989) acknowledges that as project conditions vary, so it might be necessary to carry out each section of the method in parallel, whereas on other occasions a purely sequential staged development is called for.

> "To deal with the individual nature of any project, the project manager must be prepared to modify his approach midstream, if necessary."

From this a developer/project manager can see that the structure behind the method will become blurred under certain circumstances. As the structure becomes less rigid, inherent control also disappears.

The additional difficulty of trying to cope with modification is one of the main reasons that prototyping was conceived. It suggests that the control of the process is not determined rigidly by the method but by the application of the developers experience, or some other controlling measure to a particular set of circumstances. The actual method itself is perhaps irrelevant to control of the project being either good or bad.

Furthermore, Ince (1991) states that:

> "...However, what is interesting about SSADM and LSDM (*a similar development method*) is that very few developers use all their facilities. Most of them tend to use about 60 per cent of the notations and processes."

This approach will also lead to deviations from the standard method and hence difficulties of different interpretations arising. If a method is comparatively straightforward, and can be applied to most sets of circumstances in a flexible manner, it is likely that more developers will use it in a consistent manner. Lehman (1989) suggests that an organisation will gain maximum benefit if the same general process is used throughout an entire organisation, with instantiations to suit particular circumstances.

5.2.2 Comparison of control: traditional methods versus prototyping

The use of a structured method will involve considerable effort on building up an abstract representation of a system using a mixture of diagrams and text. However working this way is not considered to be the most natural way of solving problems. Hartson and Smith (1991) state that:

> "Working from concrete to abstract is the way humans naturally investigate new concepts and solve problems. To both users and developers, a prototype is concrete while specifications are more abstract."

When using a traditional specifying approach there is a fundamental assumption that the following criteria are all met in order to ensure the "correctness" of the final specification:

- The client has been able to describe all the requirements to the developer clearly and unambiguously.
- The developer has clearly understood those requirements.
- The developer has represented the requirements correctly.
- The client understands the notation used.
- The client understands their requirement represented as an abstraction in the notation used.
- The client and developer both have identical mental pictures of the required system.
- The developer has correctly identified the optimum point to move at each stage of the project.
- The requirements do not change during any stage of the project.

If any of the points listed are not true at the end of the requirements definition phase, the ensuing system will be built to a sub-optimal specification.

As every system developer knows it is very hard to ensure that all these points are met in every development! It is acknowledged by Yourdon that the process of developing a process picture using DFDs is iterative and that a developer will have to go through the modelling process several times.

However despite the fact that:

- Iteration is used (as in prototyping).
- The optimum point is decided by the developer, rather than by the developer and client.
- The system is developed entirely using abstraction (with considerable space for various problems).

Structured methods are viewed as techniques which can be controlled and will lead smoothly to a correctly working system without errors.

The final controlling decision on the above lies with the developers. It is their judgement that determines at what point the DFD construction should terminate and the next stage of the method should commence. The use of a particular method does not make the decision for them.

The above description indicates how structured methods use iteration, that the control is decided by the project manager (not by the method), and that the use of abstraction is almost inevitably going to lead to differences between a client and a developers perception of how a system should work.

Against this if we consider prototyping and evolutionary development, they too use iteration, a participative approach to project management and finally the users are not expected to understand systems or parts of systems at such high levels of abstraction.

The main difference between the two approaches is the aim of prototyping in encouraging iteration and some form of user participation in control. The review carried out suggests that the problem of control in software development might not be more difficult using prototyping than structured methods, but is perhaps different requiring a revised approach.

Graham (1989) considers that the use of incremental development methods are not themselves detrimental to software production, but that it is the lack of planning and control that is damaging. She states that current monolithic life-cycle approach leads to a philosophy of:

> "Never mind giving the customer anything useful, just do all the detailed specification and design first."

Project control using a structured method is usually considered to be part of "project management". In prototyping, the suggestion appears to be that because the process involves iteration, it becomes difficult to manage; this is not necessarily true. It might be because the process is iterative and we do not have appropriate experience or guidelines (or both!), that causes the difficulty of controlling projects. Hence the difficulty does not exist because the method is fundamentally flawed, but because the appropriate knowledge to use it effectively has not been developed and disseminated. This is also likely to be true when using iteration of any form as part of a project using a structured approach.

It seems that appropriate control strategies and associated frameworks for prototyping need to be developed and tested. The approach to control should be comparable with structured methods in terms of control and more sophisticated in terms of user participation and validation.

5.3 Methods for controlling prototyping

From the last Chapter it is clear that there is both anecdotal and survey evidence that suggests that using prototyping may lead to project control problems. However both prototyping and evolutionary development methods do offer significant other benefits so should be further developed. It is the way that project management should be applied that will be addressed in this section.

Mayhew and Dearnley (1990) suggest that the prototyping process requires a new approach to project planning compared to more established methods:

> "It is clear that project management for prototyping must itself be more dynamic: a fixed period progress review is no longer applicable. This is undoubtedly an area that requires further research."

One method advocated is the change classification approach proposed by themselves in 1989 (Mayhew *et al* 1989). Other workers in the area also recognise the difficulty of dealing with control. Janson and Hammerschimidt (1990) state:

> "The issue of control should be raised in connection with determining the number of prototype iterations and in deciding the ultimate fate of the prototype. The question here is whether the user or the system developer decides if additional prototype iterations are unwarranted and whether the prototype will be discarded or implemented as the final system."

Both of the above references illustrate clearly that the problem of control is acknowledged and that research is required in this area. However this seems to be in contrast to work concerned with structured methods which does not appear to regard control as a particular problem area.

One way of controlling the prototyping process is to limit prototyping to a particular part of the software development "cycle". This use of prototyping might involve another "global" software development method where the project management and control is derived through adherence to the global structures and guidelines.

This type of method is typified by "Structured Prototyping" advocated by Shoval and Pliskin (1988) - see Chapter 2. It uses prototyping as a verification process for analysis carried out using the ADISSA method. It is stated that prototyping is not a substitute for structured development, but that the results of structured analysis are used to produce the prototype. The prototype is used as the sole communication tool with the users, rather than any other products of the analysis.

The adoption of a more limited use of prototyping does not in itself guarantee that control is automatically enhanced. The use of prototypes rather than DFDs or any other form of diagrammatic representation is certainly likely to improve the communication between developers and users, but using a hybrid of prototyping and structured methods does not itself, automatically guarantee more effective control.

Several modern structured methods also advocate the use of prototyping as part of the development process. SSADM version 4 now recommends that prototyping is used as "Specification Prototyping". It states that it has its own prescribed place (Step 350)

and that particular procedures are associated with it. In order to carry out this phase 7 inputs are required (Eva 1992):

- Data Catalogue
- I/O Structures
- Installation Style Guide
- Prototyping Scope
- Requirements Catalogue
- User Role/Function Matrix

The stated role of prototyping in this context is to discover errors and identify new requirements.

The author contends that using prototyping is a valid approach to successful requirements analysis, but that the above list required is too demanding in its scope and does not allow for prototyping earlier on in the development.

The examples given above both use prototyping as a validation tool, however it is not clear quite how the prototyping itself or the changes introduced into the system are monitored and controlled. The management of change and hence the iteration still remains as an issue.

5.3.1 Published methods of controlling prototyping

The change classification approach (CCA)

Mayhew, Worsley and Dearnley (1989) stated that at the time of writing:

> "No indications have been found as to how best to control the actual process of prototyping"

The publication then documents a change control strategy which aims to increase the control that project managers have over a prototyping project. In particular it seeks to help in the following areas:

- Is the prototyping making progress?
- Are the costs outweighing the benefits?
- What affect on the whole project are the alterations likely to have?

The method they propose (and then evaluate in a case study) categorises change into three classes.

Cosmetic Changes: These are trivial that have no additional repercussions. Typically they comprise changes to screen layouts, reordering fields on reports and font or type changes.

Local Changes: These have a limited impact on the system which could comprise items of the following type: adding or removing fields from a report, modifying fields on screens or reports, modifying information so it appears in a different format.

Global Changes: These are changes that may well require the introduction of new requirements and significant design changes to the system.

By using this strategy the authors suggest that developers can assess the impact of all changes required. It is important that a set of procedures are in place that define the actions required for each class of change. By recording and classifying every change request project management decisions can be taken in an informed and controlled manner. It is suggested that trivial changes can be incorporated directly into the prototype, local changes can also be implemented without delay, although back ups of previous versions must be kept. Global changes should trigger major reviews which will assess whether or not the change should be implemented.

The experience of the authors using this method is positive. They comment on the necessity of having control procedures in place prior to prototyping in order that project control is maximised. They expressed some concern that although the process of prototyping always converged it could be possible that this would not always be the case and that further work is required in this area.

This approach to change control represents one of the first serious attempts to manage iteration and change in a practical fashion. Although the authors do express some misgivings about the issue of convergence the use of this method illustrated some major flaws in a previously constructed "fixed" logical design and allowed iteration involving prototyping to be organised and reviewed in a pragmatic fashion. Brown *et al* (1988) stated that very few designs for major new systems have been successful without an extensive redesign which suggests that the notion of fixed logical designs constructed without prototyping are likely to contain some errors or incorrect assumptions. This view is given further weight be the contention of Graham (1992) that:

> "A frozen specification can only hope to meet a frozen need."

5.3.2 Extensions to the change classification approach

Williams (1992) proposed an extension of the CCA based in order to try and capture both the scope and impact of proposed changes. He justifies this by consideration of the following question:

> "What is the classification of a change which is trivial, and yet affects a part of the system other than that being prototyped ?"

He found that assigning classifications to a local government system proved difficult particularly where:

- There was a problem differentiating between a local and global change
- A change required significant work, but affected only a small part of the system or current prototype.

He illustrates this with an example of a change which took six minutes to implement but the scope of which fell outside the system being developed. Hence it was difficult to categorise as either trivial or super-global.

As a result of this he suggests a revised classification scheme which attempts to overcome this problem. This is shown below:

Local (Trivial/ Small/ Large)

Global	(Trivial/ Small/ Large)
Super Global	(Trivial/ Small/ Large)

He suggested that this classification scheme will provide more accurate classification and impact analysis for project managers.

Davies (1993) also introduce a derivation of the CCA and includes extra management control procedures that he considers to be important. He suggests that the main change control categories should be as follows:

Cosmetic	(No impact indicators)
Local	(Small |Medium |Large)
Global	(Small |Medium |Large)
Super Global	(Small |Medium |Large)

He contends that the cosmetic classification should be left in as it "unambiguously" identifies a change that can be implemented with a minimum of effort. He also uses different terms within the impact analysis classification as he feels this gives a better spread. He concurs with Mayhew (1989) that there need to be appropriate management procedures in place to ensure that the change control mechanism can work effectively.

In addition to this framework he suggests that additional controls are required and suggests extra mechanisms:

- Change classification (as outlined above)
- Use of function point analysis for estimation and productivity monitoring[1]
- Timebox control - The time period during which development takes place prior to reviews.

[1]The reader is directed to Albrect and Gaffney (1983), Symons (1988) and (1991) for an explanation of the theories behind Mark 1 and Mark 2 function point analysis.

- Function priority - The agreement of users and developers of level of priority of requested functions
- Maximum cost
- Version control
- Prototyping session control - Procedures for making most effective use of prototyping sessions
- User sign off - Agreement signed off by users.
- Planning and recording of work effort - Ideally through the use of software tools.

The author evaluates the use of these mechanisms in the development of a system for a local education authority and makes a set of observations based on the work carried out.

There were serious problems using function point analysis when estimating the work required for a given change. Any changes requiring work which decreased the complexity of the system had a misleading effect on the productivity/effort figures if used in their raw state. It was concluded that the use of function points alone was not a sufficiently useful or accurate form of measurement. The author commented that the study of metrics for use in controlling evolutionary development required further work.

The change classification approach was very useful and the rate of requested change dropped from an initial high level to a much lower level as the development progressed. This was regarded as positive evidence of the use of the method.

Testing created difficulties in that it proved very difficult to evaluate the effects of small changes, particularly where complex interface issues were involved. The author considers that this form of development requires further investigation with regard to the issues of testing.

Although Davies highlights some problems with the development, it is clear from the work that both developers and users were very pleased with the results that were produced.

5.3.2 The DSDM approach to project management, control and estimating

The DSDM approach to overall control and managing iteration is only outlined at a very high level. The approach suggests that there are three important development aspects that must be managed to achieve a successful path through the prototyping cycles. These are as follows:

- Change control - It recommends that the procedures for change control must enable changes to be introduced quickly and effectively.

- Configuration management - It suggests that configuration management must be controlled so that backtracking is at all times easily possible.

- User involvement - Users must be truly representative and must be available at all time in order that the project progresses evenly and does not slow down.

The DSDM approach to controlling iteration is also only considered at a high level. It is suggested that three prototypes (investigative, refining, and consolidating) should lead to an acceptable functional model, and that up to a further three for the implementation. It also suggested that this might be reduced if the modelling and build of the system merge together.

Estimating with DSDM concentrates on those issues which it considers are fundamental to the method rather than on guidance on how to produce estimates. It supports both the use of function points and timeboxing and that both should be monitored more frequently than with conventional projects as the speed of development is faster. It is suggested that the use of function points is a good way of allying estimation to the high level business functions that are required to be supported. However apart from some very general guidelines about recording estimated and actual function point counts little guidance is given to guide a prospective user of the method.

5.4 Discussion

There is evidence from published work citing case studies (Mayhew 1989, Williams 1992, Davies 1993) that shows with some confidence that using change control in conjunction with appropriate project control strategies can lead to software being delivered on time with a high degree of user satisfaction. The DSDM approach to

project control is very high level and that without evidence of actual effectiveness (i.e. via published work) the usefulness of such measures must remain debatable. In particular the author questions the use of function point analysis as a useful predictor of system complexity when evidence from Davies (1993) casts doubt on its value. This point is examined later on in Chapter 8.

Part 2. Action Research.

The work presented here uses action research to examine the use of prototyping in commercial software development projects. Two separate studies are examined in one organisation plus a further study in another. In each instance a significant amount of the work carried out has involved prototyping. The aim of this section is to evaluate how prototyping could be used in a variety of ways to help aspects of systems development.

There is an introduction to a systems development problem in a metal finishing company with the development of an information system for use within the company. The use of interface prototyping within the project is outlined.

A further study is documented on a decision support/hybrid expert system development for the same company using different development tools and methods.

A separate study in the use of function point measurement during the development of a system for a local authority is reported.

Some provisional conclusions on the use of particular prototyping methods and techniques are discussed.

Chapter 6. Initial Use of Prototyping

6.1 Aims and Objectives

The aim of this part of the research work was to try and discover whether prototyping could be used as an effective aid to systems development. It was introduced to the target organisation (Frost Electroplating of Birmingham, UK) in order to investigate this. The systems development work had been initiated using a more traditional approach and it was felt that the adoption of certain prototyping techniques would be beneficial. A more detailed background to the organisation and the problem under investigation is documented in Appendix 1[1].

It was recognised that the use of prototyping could perhaps help to provide a clearer definition of project boundaries and scope so that everyone involved could get a better view of where the project stood and where it was heading. Hence it was intended to help formulate system requirements by stimulating a joint learning process between users and developers.

In prototyping users can be involved from the conceptual design through to the finished model. It was hoped to use this approach to try and resolve some of the conflict or ambiguities that had arisen from different groups expecting or requesting different system requirements.

It has also been demonstrated that prototyping should be useful as a communication tool between the users and the developers (See Chapter 5). Hence this technique was used for requirement elicitation and computer based prototypes of how the eventual system could operate were built. However, two important attributes must be appreciated:

- It was not intended to build every aspect of the system, so it was not considered appropriate to prototype every single screen and function

- The prototypes would not necessarily be created with the intention that they became the final system, although ultimately in the first exercise (the "interim" system) this did prove to be the case.

[1]The appendix gives a detailed description of the analysis that was undertaken in developing a costing and estimating system for a metal finishing company.

It was felt that the difficulties that had arisen during the first analysis stages of this project could perhaps be solved by using prototyping to try and develop parts of the system in separate stages. It was decided to use prototyping to help the users try to define the final specification they required the system to have; something that had been impossible to explain using traditional analysis techniques alone.
In order to try to discover more about the costing and estimating problem it was decided to split the investigation into two distinct sections and to use prototyping as the main development process. The two sections comprised:

- Interface and incremental prototyping. Screen designs, report layouts would be constructed for the purpose of designing an "interim" system (which in itself would be delivered in stages) and defining user requirements more precisely. (This is dealt with here in Chapter 6).

- Functional and evolutionary prototyping would then be used to discover the parameters that control cost and price and would be employed in implementing the final solution. (This is dealt with in detail in Chapter 7).

By adopting prototyping during this part of the investigation it was hoped to investigate whether the following aims could be achieved:

- To allow an early partial payback on the investment

- To give a partial solution which could nonetheless go some way to solving some of the existing problems of pricing and estimating

- To allow staff familiarisation with a computerised pricing systems and to produce user acceptable input screens

- To focus the attention of the estimators on the most important factors considered during costing and estimating and increase user/developer communication. I.e. to consolidate understanding of determinants;

- To get a clear definition of the system requirements and to help identify the boundaries of the system

The last aim here requires further explanation. The users were not clear about the level of expertise that the system would exhibit in automating pricing; at this point in the investigation they had not been finalised.

6.2 Outline of system requirements required by the organisation

In addition to the aims of the research project, there were also requirements that the organisation themselves had. It was recognised that a computer system should be capable of preventing or solving most of the following identified problems.

- Better levels of consistency within the estimating system were required. All prices should be capable of amendment but the estimator using the system should be able to produce a price that has a known profit margin associated with it. Management could then direct whether the price is to be used with or without amendment according to conditions prevailing at the time.

- Quotations needed to be indexed, and then accessed on the system. This saves a considerable amount of time spent trying to find (manually) copies of quotations which are thought to be relevant. Finish code, customer account number, and quotation reference number or similar criteria could all be used to provide a good cross referenced indexing system.

- Quotations could be produced on the system. A standard form layout with specific information fields would be designed. Its design should facilitate fast data entry on the terminal and print options that would be able to produce easy to read quotations in a short space of time. The use of the computer would also allow amendments to be made quickly without the need for whole documents to be duplicated.

- The system should be designed so that it can automatically carry out many of the calculations currently done manually. This would help remove the inconsistencies and errors that arise from prices calculated manually. This problem was compounded by the combination of metric and imperial measurements that were used by the estimating staff which created extra complexity.

- Amendments to metal prices, and other costs associated with producing an accurate estimate could be made quickly. This would allow easy comparisons to be made between prices produced at different times if necessary.

If a suitable system could be developed that incorporated the above characteristics the need to have an "expert" estimator available to do every job might be avoided. This would allow them to spend more time on those estimates which required most analysis. The computer could be utilised to produce those quotations which were straightforward and did not require the levels of judgement that only a true expert can be expected to bring to bear on the problem. This type of system might not be able to emulate the entire knowledge of the expert estimator, however even a simple decision support system could aid in this area.

The final system was intended to lead to a solution that managed to significantly improve the consistency, and reliability of the overall estimating process. This would be combined with a significant reduction in time preparing the quotations. It was not the intention that the system would take over from an existing expert but would act as a decision support tool to help produce quotations quicker and more consistently. In addition it is important to note that the proposed pricing and estimating system did not attempt to duplicate the current manual procedures in use at the factory. It was not the intention of the research to design a system to achieve this. However it was of course possible that the automated system might appear to mimic some of the current costing and estimating practices.

Eventually the final aim after all the stages of the investigation was to create a specification that included the following features:

- The facility to generate prices for all or most jobs received for tender.
- A choice of levels of machine/expert estimator interaction.
- Advice on likely minimum profitable prices to charge.
- Easy adaptability to incorporate management decisions on costing and its affect on pricing; the system must be designed to avoid complex adjustment to suit cost/policy changes.

It was hoped that using prototyping would allow this to be facilitated.

The Yourdon type analysis that took place revealed a variety of shortcomings with the present system of producing quotations. The use of levelling and decomposition of the problem into well defined processes and procedures could not be easily accomplished.

Hence it was impossible to construct a valid specification that could be turned into any form of working system that would be useful, the information gathered was insufficient. It had been intended to use the information provided by the pricing and estimating staff as a starting point for a full computerised system. The unsuitability of the information, however, precluded this. The data flow diagrams which were produced after lengthy consultation with the estimating staff could not be broken down to such a level that a specification could be produced and coded. This is illustrated in figure 6.1.

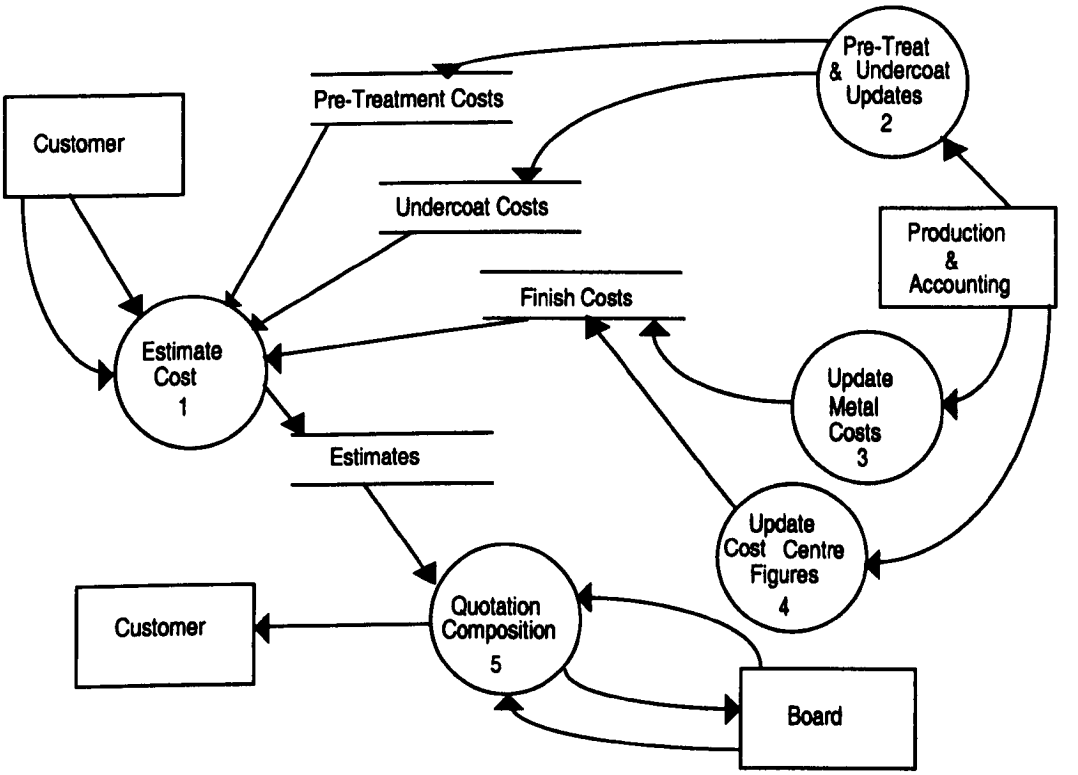


Figure 6.1. The main processes in the costing and estimating system[2]

The process recorded as number 1 (Estimate Cost) could not be decomposed to a level which allowed a process specification to be written and hence implemented. The staff involved were not able to adequately describe their mechanisms for carrying this out in sufficient clarity for the development team to take this further.

It is interesting to note that the Alvey programme recommended that design by rapid prototyping with the interface as the critical item should be used in preference to life-cycle or user-centred design methodologies. (Klein and Newman 1987). It was felt that this technique could be adopted to try and overcome some of the problems that had arisen.

[2] The data flow names have been omitted for reasons of clarity.

The initial interface prototyping stage was intended to create a system that would appear similar to a likely final version, but would not contain any pricing calculations. The estimating and pricing would still be calculated in the previous manual manner, but the computer would allow certain processes to be automated. These processes included:

- Quotations could be "written" using a standardised form held on the computer.
- Report generation was made possible.
- Record retrieval was facilitated; quotations were stored to allow access using different search criteria e.g. finish code, date, customer code, and quotation reference number.
- Easy amendment and deletion of quotations.
- Quotations were designed to allow information in addition to that required by the customer to be held. (This was to create a file of information about a part which could be easily transferred to the company parts file if the tender for the work was successful).

The front end of the system was designed during this stage. It had the appearance, (i.e. all the input/output screens and reporting capabilities) of those features which were anticipated to be present in the final system. However it lacked functions which it was anticipated the final version would ultimately have.

The quotation design provided a form layout that included factors that were likely to contribute to the final price. Inputs to a screen were stored so that a quotation document could be produced, then printed and sent to a potential customer.

By leaving out calculations from the program the main responsibilities of the costing and estimating staff remained the same. The manual methods of price calculations were carried out as before. Although it was not to be used in any machine based calculations, the staff were encouraged to fill in the data that they used whilst making the calculation. After a while in operation some basic arithmetic operations were added to the initial version at the request of the principal users.

After the system had been operational for a while it was intended that the extra information provided would allow, during the second stage, progress towards a fully

automated estimating and pricing scheme. The original intention was to make this an extension of the "interim" system produced by the initial prototyping phase.
As the traditional method employed had so far appeared to be difficult to use to effectively analyse the problem it was felt that using user interface prototyping could provide a more effective method for eliciting requirements.

An incremental approach was adopted with interfaces initially being developed on the target machine and then implemented as working pieces of software in stages. This gave the company a working system for quotation production, modification and retrieval, and hence a faster pay back on some of their investment.

One of the stated advantages of prototyping is to allow a fast payback on the investment in a project. By adopting an incremental prototyping approach it was intended to build a partial solution to the problem in a short space of time. The effort expended in the requirements analysis took several months and the clients were expressing concern at the progress made. This was certainly partly due to the inexperience of the researcher in the area of the research but was also due to the complexity of the problem and the difficulty found analysing it using a structured technique. By using prototyping it was hoped to minimise the time spent getting a system implemented that could be used.

Eventually it was hoped that by extending this into a more evolutionary approach the system could be further extended and improved as a comprehensive solution to the problem was discovered.

The analysis carried out highlighted areas where a computer system could speed up existing procedures and help the estimators carry out their roles more effectively without calculating the final prices. By implementing a partial solution, less time would be required in designing and coding the final solution. It was felt that the effort required to produce a comprehensive specification for a complete solution would be very considerable, particularly as the effort on analysis had so far not been a successful exercise.

The estimators were not familiar with computerised costing or pricing systems. Although they had experience of computerisation of other application areas (e.g. existing parts file listings, customer records, and order processing), it was felt that experience with some form of computerised costing or pricing system would allow better user and developer communication during the rest of the project. It is essential that the feedback from users is taken account of during the system design. Slatter *et*

al(1988) noted the problems of ignoring the reactions of users to a prototype estimating system:

> "this indicated to the project design team that the end-users had not had sufficient influence on the design of the prototype system."

It was intended that the experience of using the interim system would help the estimators to elucidate the methods that they used when devising prices for a quotation. The use of the interim system would be monitored. It was hoped that the use of standardised quotation form layouts and the input of considerable data about the nature of each quotation would allow further analysis of criteria considered when the estimators were considering the calculation of prices. Further prototyping would allow a more comprehensive solution to be drawn up.

6.3 Carrying out interface prototyping

The computer in use had the facility to allow screen designs to be built without other parts of the program requiring coding. It was decided to use this facility to produce the screens that the users would input data to and receive output from. Although some research suggests that screen designs and prototyping in general is best executed using "high productivity" code generators and fourth generation languages (see Chapter 4) it was felt these basic screen painting facilities could help to prototype some parts of the intended system.

As the intention was to implement the final system on the existing computing facilities at the company it was felt that using any software tools present on that machine could speed up the design process.

Sketches of proposed layouts were produced which were discussed and modified. From these, the execution of the screen designs were carried out. The designs were produced from basic specifications agreed upon at meetings with company management and the estimators as part of the high level specifications. The specifications were of a very general nature and were not of a form that could be translated into a working system.

The author was able to present the various completed screens (which included commands for report generation, or further processing as part of the layouts) to the users as part of the proposed system. See figure 6.2 for an example of an early screen mock-up that was presented to potential users of the system. The example illustrated is

the menu that forms the opening of the quotation production system and is one of the more straightforward from the work!

The author was aware of various screen standards that existed within the existing suite of programs that were implemented at the company. Some of these were derived from standard IBM operating procedures, whilst others had been developed by third party software suppliers. Unfortunately there were inconsistencies in the standards and the interface prototyping was used help resolve these and to build screens in line with exactly how the users wanted the system to perform.

This process although requiring several modifications of the original ideas allowed the users and designer to agree on the system as it was being produced. The process of consultation ensured that if any areas of disagreement, or misinterpretation of specification occurred it was amended or rectified before a substantial quantity of extra work was carried out.

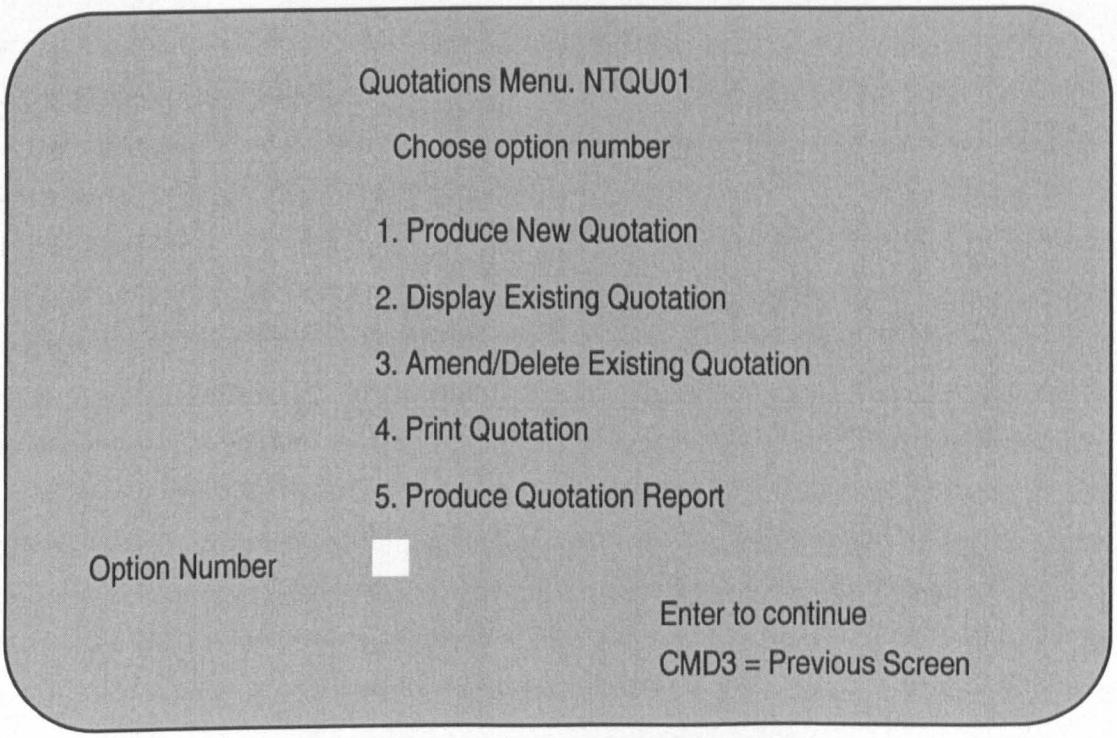


Figure 6.2. Example of screen shown to target users

The project was regularly scrutinised as it progressed. The constant review process also allowed relatively easy amendment to the first agreed specification.

It was decided that an extra information file broadly in line with an existing factory parts file would be part of the first system. This occurred after the screen design process had commenced. However, although the initial specification was affected by this

decision, the effect on the work already carried out on the existing parts of the system was minimised as this modification was agreed on substantially before the end of the development work.

First attempts at specifying the (initial) system had concentrated on written specification proposals combined with printed screen designs. The response of the users to this method of specification representation compared to using actual screen designs was not as positive. Users felt more confident about commenting on actual screens than on drawn representations. The screens could be linked in such a way as to show possible sequences of processing. This proved to be a useful technique as users could follow suggested ways through the system in a logical sequence.

Hekmatpour and Ince (1986) cite various means by which a system design can benefit by the use of prototyping. The work carried out shows that many of the points raised concerning the use of prototypes were relevant to this study. The use of screen designs, particularly when presented in the correct sequence, is a major benefit to user/designer communication. The user can see exactly how a system will appear and seems to react favourably to this method of proposed system design.

The users appeared to be more willing to criticise proposed specifications when the amount of drawn designs and written documentation were kept to a minimum. The concept of prototyping is designed to help overcome several problems that exist with "conventional" specification documents. The original methods at the organisation used to help decide on specification did have problems. The customer requirements were misinterpreted, poorly understood and required change as the specification was being developed.

The use of screen generators and careful sequencing creates the impression of a working system that can be easily understood by the user. The external view of the system then dictates the internal (processing) components. This type of approach reverses the traditional type of specification design with its emphasis on process rather than appearance.

It is very important for the designer to accept criticism during the prototyping process as this leads to increased co-operation with the users. Unless the designer is prepared to accept changes which he or she may not agree with, the users may well end up with a system that has been dictated to them and will not perform as they require.

The work carried out during this part of the project tended to follow the theme of incremental or even evolutionary prototyping where software is developed and modified to adapt to the changes as they occur, rather than fixing the specification and then building an entire system from that point.

This (initial) part of the system was developed in 3 separate phases. Each phase produced a sub-set of working software that had been developed in collaboration with the appropriate user groups. Hence although the initial overall development was considered to be mainly an exercise in interface prototyping (eventually leading on to an evolutionary approach), the implementation of the first part of the system had more in common with an incremental approach. The author feels that the dividing line between these two definitions is blurred and to some extent artificial. This is in broad agreement with some of the work reviewed in Chapter 3.

6.4 Difficulties encountered during interface prototyping

Although the interface and incremental prototyping produced a system that was tested and in use in a comparatively short space of time (of the order of 6 months for all three parts), there were some problems in its development.

There was a problem regarding the overall control of the project. The interim system allowed much faster access to quotation records, extensive reporting facilities and faster production of standardised quotations. However the final implementation was delayed because of conflict within the user group as to the most appropriate style of quotation that should be sent to a customer. Although the programming required to change printed output is not complex, the system could not be properly tested whilst disagreement between users existed. As well as postponing the final implementation of the system, it also delayed the start of the next stage of the investigation. Prototyping although actively encouraging iteration still requires tight project control to work, it cannot be effective without it. This experience is very much in line with the problems alluded to by respondents to the surveys reviewed in Chapter 4. It was felt that the overall purpose of the project and its direction were allowed to drift during this phase. At the time this caused the problems listed above and unfortunately it was difficult to resolve the conflicts because of the users' respective ranks within the company. Although the user team had been carefully selected with a senior company member as the main decision maker, senior personnel not directly involved in the project created these project difficulties.

The actual prototyping process should have good records of all the sessions carried out, and all the changes suggested and implemented as a result. It is possible where misinterpretation has occurred that the developer might make a mistake in an updated version of the prototype. Each prototype should be recorded and all modifications noted. If necessary it should then be possible to return to an earlier version without difficulty, thus avoiding delays in trying to recall earlier versions. This process does take time, but the consequences of not carrying it out thoroughly can be serious.

The development of the interim system on the IBM System 36 allowed quick change to screen designs following prototyping sessions with the users. Once more functionality started to be built into the system the developer was required to learn the RPG2 programming language. This is not a quick process as the language is fairly complex and requires a certain level of knowledge before any useful tasks can be carried out. This had two effects:

- The project progressed fairly slowly after a good start whilst the developer became familiar with the language.

- The final system could eventually be written more quickly (if the developer was going to implement the eventual specification) as the developer would then be familiar with the programming language and style of the programming.

In prototyping there will be a certain degree of "front loading" at the start of a project if the developers are required to learn a new language or technique. This will be combined with considerable time expended in communication with users during the development of prototypes, it is important that clients must be made aware of the time required at the start of the exercise. Harker (1988) notes that:

> "there is a requirement to allow for both time and resources to be devoted to the prototyping activities in the earlier stages."

Prototyping imposed a heavy burden on the users. If they are not prepared to co-operate in the process, prototyping is unlikely to be successful. This experience is very much in line with the results from the Kinmond (1994) survey work carried out.

If the prototyping is intended to produce a discardable prototype then the choice of tool should be influenced by the ease and speed of use of the prototyping tool and should try

to avoid techniques which require large amounts of time to learn or to change prototypes.

The difficulties of interface prototyping and the need for designers to be both skilled programmers as well as have expertise in human factors is an area that has been recognised in research work (E.g. Gray *et al* 1988). It is important to analyse requests so that badly designed interfaces do not become part of implemented systems.

The "interim" system built as a result of the interface prototyping was given a trial of several months in the factory. Initially there were some operational errors in the software that required correction although these were not considered to have had a seriously detrimental effect on the overall project. The system is now being used by the estimators for the production of all quotations, facilitating a more streamlined operation. The data captured was utilised in further research for comparison with prices produced by prototypes.

The use of interim "prototypes" as systems does require careful planning. There is no doubt that the functions of the system implemented were developed from an exercise which was only initially considered to be interface prototyping. By eliciting the requirements from users, using only screen designs and report layouts the final system design was developed and implemented. Hence this type of exercise raises the question of whether it is feasible to use interface prototyping in isolation without considering major aspects of function.

6.5 Summary and conclusions from using interface prototyping

The outcome of whether the aims of this particular piece of research were met is detailed below, the aims (in italics) are duplicated from earlier in this Chapter.

- *To allow an early payback on the investment*

- *To give a partial solution which could nonetheless go some way to solving some of the existing problems of pricing and estimating*

The experience within this project suggests that an early payback can be achieved using prototyping, particularly if an incremental approach is adopted by the developers and that this is accepted by the users of the system. From a psychological view it seems that

using a highly interactive systems development method can retain the focus of the users and maintain their interest in the project.

- *To allow staff familiarisation with a computerised pricing systems and to produce user acceptable input screens*

The system developed required a significant amount of change during its development. This suggests that by adopting prototyping and provided that feedback from users is treated correctly, a system produced should allow a high degree of user acceptability. It was sensed that the main users did develop a sense of ownership of the system which certainly aided the acceptability of the eventual system.

- *To focus the attention of the estimators on the most important factors considered during costing and estimating and increase user/developer communication. I.e. to consolidate understanding of determinants;*

- *To get a clear definition of the system requirements and to help identify the boundaries of the system*

The focus of attention of the users did depend to a certain extent on their level of co-operation. Not all users react in the same way, and hence determining requirements accurately and defining clear boundaries to the system are governed to a large extent by the input from users. The level of developer/user communication was certainly high, but on occasions it was felt that with one particular user the usefulness of the information received was sometimes lacking. Hence this experience suggests that if a project does run in to problems regarding the quality of communication within it, then it could well suffer serious consequences. This could lead to problems with regard to accurate definitions of requirements and problems of defining system boundaries.

However the first exercise in prototyping using interfaces as the main focuses of attention on the system specification was extremely worthwhile. The system produced was tested in a limited area of the factory and is now being used for the production of all the quotations. It has significantly speeded up the quotation process and has required virtually no amendment from its original implemented version (apart from minor bug fixes).

6.6 Further issues raised

The actual prototyping revealed several points which required attention prior to the next part of the project taking place:

- Better project management was required.
- A formal change control strategy should be adopted.
- The prototyping tool used should be carefully matched to the type of prototyping and the type of system under development.
- The relationship between prototypes produced and eventual specification was not clear, this remained a problem.

As a result of the experiences it was decided to research further into, and give further thought to, the management of prototyping before the next part of the exercise. The relationship between prototypes, documentation and specification was not clear. Ince (1988) suggests a prototyping plan where:

> "The prototyping objectives, the prototyping model adopted, a description of how prototyping is to be achieved, and the use that is to be made of the prototype, should all be written into a document known as the prototyping plan."

Analysis of the outcome of interface prototyping and a review with senior management led to a change of system boundary. The project was no longer trying to produce a fully automated pricing system. Instead an automated costing system that would provide a consistent base for manual pricing was adopted. Interface prototyping had enabled both developers and users to get a better understanding of the type of problem being investigated. This was noted by other authors, although some researchers do go further, e.g. Buckley *et al* (1988) put forward the idea that the major purpose of interface prototyping is to gain insight into functional requirements.

From this experience the author feels that there is not a clear cut divide between the domain of interface and functional prototyping, although it is important that developers set out with at least some clearly expressed ideas about what they are trying to find out about. Simply designing some interfaces in the hope that functional

requirements will automatically appear is unlikely to be a successful exercise and is likely to waste time compared to a better organised and more focused operation[3].

[3]Comments on the general use of action research as a research tool are included in the conclusions of section 2 of the thesis.

Chapter 7. Functional prototyping

7.1 Aims and objectives

As a result of the work carried and reported in Chapter 6 a decision was made to proceed with further prototyping in the cost estimation area.

The main aim of this action research was to try and discover if project control of systems development using prototyping could be improved. In particular more rigorous project management and control mechanisms were put in place. The following processes were instigated in order to try and investigate this:

- Keep extensive written records of each iteration (and prototyping sessions) to ascertain how much this could help improve the project control;

- Use more than one developer at prototyping sessions to see what effect this had on the sessions carried out;

- Use of a change control strategy to try and improve overall project management.

From the experience of using interface prototyping with incremental delivery and a thorough analysis of the information gathered from the use of the structured techniques it was felt that an exercise that concentrated heavily on the rules and procedures that governed three separate problems should be addressed. These were considered to be essential to a consistent costing and pricing process.

For costing:

- Identifying the 'route' through the factory a batch of components would be likely to take and hence the processes that would be applied to them.

- Applying an up-to-date cost on a 'route'.

To extend into the area of pricing:

- Manually adjusting a cost to yield a price.

The 'route' through the factory is particularly significant because costs associated with apparently similar components can vary by up to a factor of ten. Provided a subsystem could be developed to assist in the accurate prediction of 'route' and another subsystem to maintain the cost of routes, then the cost estimation problem could be solved.

The route that a job would take through the factory determines what costs are incurred as it is finished. At the estimation stage it is impossible to determine with perfect accuracy exactly which process, and for how long that process will be applied unless the job is identical to one that has been carried out before. The inconsistencies in the pricing process could be reduced if a system could be designed that would allow the determination of route to be more accurately predicted.

By applying costs to the processes (from the company's current costing figures) it was thought that it should be possible to get a good indication of the costs that are being incurred by each job as it passes through the factory. The decision on the final price to be charged will remain with the estimator, but the type of system considered should allow them to make a better judgement of the likely profit that a particular job should make. The adjustment of costs to give prices is both complex and dynamic depending upon market forces. In some cases the actual adjustment is applied by a member of the board. It was felt that it would not be practical to try and extend automation into this area.

The experience of the first prototyping exercise had been positive and had led to a better form of communication between the users and the developer. It was felt that in the light of the experience gained it was worth extending the prototyping into the second part of the specification design using functional prototyping.

The literature states that this method of requirements engineering needs a considerable input from the user. The prototyping sessions were set up to try to discover and clarify the functionality of the system. The presentation of the information was not fixed and was modified to suit the user as they preferred.

7.2 Choosing an appropriate prototyping tool

The nature of the problem suggested to the developers that the problem might best be considered initially by using a rule based system. The overall requirements had been discussed and it was felt that the system was eventually going to be some form of decision support tool based on rules. The IBM System 36 installed at the factory did not

have the facilities required to support a prototype using rules which could be modified quickly, so it was necessary to consider software tools that would run on a PC.

After the experience of the interface prototyping exercise it was decided that the highly iterative nature of the prototyping process could be speeded up if the development time required to make changes to prototypes could be improved. Additionally it was decided that a system that could be quickly learnt would also reduce the time overheads incurred at the start of the investigation.

After examining several alternative possible software tools to use it was decided to use the expert system shell Crystal. It was chosen because:

- The evidence gathered so far suggested that a rule based approach was likely to be appropriate.
- It was straightforward to learn.
- It was very quick to modify.
- It had a clear user interface.
- It could be interfaced with other software if necessary.

During the course of the investigation an upgrade was announced for the software. This caused a slight delay as the developer familiarised himself with new features offered, but did not cause a significant interruption in the development process.

Because the tool was chosen with the intention of building prototypes to discover further requirements, it was decided that the prototyping would be of a throw-it-away nature. The final goal was to draw up a suitable specification that could be used to enhance the software that had already been installed on the IBM system. The prototypes developed in the expert system shell could form part of the eventual specification but would not be used as an implemented system as this stage of the development was carried out using a PC rather than the IBM System 36.

The aim was not to construct an expert system per se (which is the type of system Crystal is designed to build), but to investigate further the decisions taken when routing

a job through the factory. The exercise would start purely as an attempt to discover functional requirements.

7.3 The choice of area and personnel within the factory

It was felt that the prototyping should not try to discover all the decisions taken to predict all the routes in the factory. By trying to model a sub-section of the factory it was hoped to discover how practical this type of approach would be.

The choices of area and personnel were considered carefully. The following requirements were thought to be necessary to make the prototyping as effective as possible.

- The factory area should be fairly representative of the factory as a whole (if possible).
- The costing figures should be available.
- The personnel should be available.
- The personnel should be amenable to the exercise and if chosen would co-operate with the research.

Although some of the requirements listed above appear to be nothing more than common sense it is important to realise the implications of any of the above not being fulfilled. This is stressed in the literature (e.g. Mayhew 1990, DSDM 1995b).

By choosing a representative area of the factory for the route model, it was hoped that a fairly confident prediction of whether an extension of the prototype into the whole factory would be successful. An area with a very limited range of processes would give a falsely trivial view of how difficult the problem was likely to be.

It was important that recent costing figures were available in order to allow a comparison of predicted costs against actual costs incurred. Only by thorough testing of prototypes is it possible to make a valid judgement of how complete and useful they actually are.

The choice of personnel is also important. The users involved should be able to make constructive criticism of a prototype such that any changes suggested can be interpreted by the developer and incorporated into the model. It is important that they realise that only by their criticism will a prototype be improved and refined. The communication between the user and developer is the key to the likely success of the prototyping exercise.

After consultation with the management at the factory and discussion concerning the production areas that could be used in the study it was decided to use the silver production area. It was considered to be most suitable because of the following reasons:

- It had a representative range of processes and production methods (Various barrel types and a rack area).
- Good costing figures were available.
- The staff were experienced and were likely to be able to articulate their comments well.

In addition to the above factors the management already used a simple form of computerised pricing for the silver department. They felt it should be possible to use this as a reference for figures suggested by the route/cost prototypes.

7.4 Conducting the prototyping sessions: using a change control strategy

The choice of strategy was inspired by the control strategy advocated by Mayhew (1989). Using this method (discussed in Chapter 4), all changes to prototypes suggested by users are recorded and classified into three categories (cosmetic, local and global) according to their impact on the system. It is recommended that both cosmetic and local changes are made immediately with the latter being subject to later scrutiny. Global changes which have an impact beyond the parts of the system currently under review are recorded but change is delayed until all the implications have been researched. Hence the developers retain control over changes which have wide reaching effects whilst still leaving users with the impression that it is their system.

7.5 Carrying out the functional prototyping

The prototype sessions tried to clarify the function of the system, not the interface. The users were made aware that the presentation of any information was not fixed and could be modified to suit their requirements. However it was recognised that the function and interface were related and could not be presented at prototyping sessions as entirely separate issues.

The information in the system was presented in a way that attempted to duplicate the decisions that an electroplater would make in the plating shop as he or she decided how to plate a job.

The specification in the problem did not exist in a well defined form. The purpose of the prototyping was to find out more about this and to examine the functionality of the system which had been designed with the limited knowledge already gained.

The implementation and the interfaces were not considered to be important at this point. Due to this it was felt to be impractical to sub-divide the definitions of change in the manner suggested by Mayhew (1988), as global/local/trivial boundaries could not be defined.

The suggested change strategy was:

- Total redesign.
- Rule Modification.
- Trivial.

This divided the changes into three sections (as suggested by Mayhew), but the categories were classified differently in order to allow the specification requirements to be better understood. Although it may be considered unnecessarily pedantic to change the categorisation in this way, it was felt that the problem would be better addressed.

7.5.1 Setting up the Sessions

The first prototyping session involved 3 people. These were:

- The silver department plating manager (the user),
- The system/prototype designer,
- The project supervisor, but acting as scribe in this case.

The session was conducted by the designer. All suggested system changes or points of note were recorded by the project supervisor.

At the start of the session the aims and objectives of the exercise were explained to the user. These were:

- The system was incomplete and required revision.
- It was designed to be used by an estimator (not an electroplater).
- The system contained information that he had supplied in the interviews that had been conducted at the factory.
- He was asked to criticise any points that were wrong. This applied to major revisions or omissions as well as any minor or trivial parts.
- He was told that the system interfaces (screen designs, colours and layouts) were not the main points under review.

Prior to the session taking place it was decided to split the exercise into three main sub-sections.

To start with, the prototype would be demonstrated, and information about sample part batches would be input. The outcome would be noted and compared against the outcome suggested by the expert. Several components of different shape, size, substrate, and batch size would be tackled in this way to see how accurately the system predicted the route and processes.

When this had been completed it was intended to repeat some of the selection, but this time to ask the expert for comments at different stages during the investigation. The sub parts of the system would then be analysed as well as the overall functionality of the complete prototype.

Finally any further general comments about the performance would be sought.

By dividing the session up in this way it was hoped to achieve 3 objectives.

- How efficient is the prototype as it is presented?
- What issues are incomplete, incorrect or missing?
- Do the ergonomics present a reasonable interface for the user?

At the first session it was not intended to address the final issue in detail as the functionality was the main topic under review. However as already stated it is very difficult to separate the function and interface entirely so any parts of the presentation that were considered unclear were also marked for amendment.

All comments were noted on a specially designed form which had been adapted from a layout suggested by Mayhew (1989).

Change Control Form. **Frost Electroplating: Costing and Estimating**

Reference: Prototype Review, silver department, route/process

Date Time Start

Finish

Reported by:

Attended by:

Function Changes Clarification

Notes:

Figure 7.1. Illustration of form used to record changes in prototyping sessions

7.5.2 Comments on the functional prototyping sessions

After each session the comments written down were transcribed into a form that summarised the results. The conclusions drawn from the particular session were also summarised.

The prototype was kept in an unaltered state. Modifications were then carried out on a new copy.

The design went through a series of iterations with the user commenting on the changes after each set of modifications were carried out. The number of changes to the system were recorded and a log of time spent was recorded.

7.6 Functional prototyping: results

Functional prototypes for the prediction of route were built with the aim of discovering the relationship between determinants of process and route. Due to the complexity of the problem initial work has been restricted to silver finishes in order to evaluate feasibility. The prototypes were constructed using a "throw-it-away" approach, the original intention being to enhance the system already installed on the IBM mini system once the requirements could be specified.

The prototyping sessions did not produce any global changes (this was expected given that the main boundaries had now been determined). Cosmetic changes were few, again not surprising as it was throw-it-away functional prototyping, rather than interface prototyping that was being carried out. It was decided that it was not practical to make functional changes during sessions; as alterations to one rule in an expert system can have a ripple affect on other rules. The strategy was to confirm the change with the user and return with an updated system quickly (normally a few days). Otherwise the suggestions made by Mayhew were found to be most appropriate. It was found that the users maintained a high level of interest and co-operated enthusiastically. This agrees with the findings of Butler and Chamberlin (1987) who stated that:

> "... the existence of prototypes was vital in keeping the lead experts' interest."

After about six sessions with users, plus three review sessions and non prototype meetings, the prototype was accepted. A summary of the time spent and the changes suggested is recorded below

Meeting No	Time (hours)	Changes Recorded
Introductory Meeting		-
Pre-Prototype Meeting		-
1	2	18
2	1.5	11
Review meeting		-
3	1.25	8
4	1	6
5	1.3	7
6	0.63	5
Totals (excluding reviews)	~8	55
Changes per hour	7	

Note: The times recorded do not include introductory and review meetings where changes were not suggested.

Table 7.1 Summary of time spent in prototyping sessions

The factory management were impressed with the outcome to the extent that they requested that it be developed into a usable system. They wanted it to be used as a sales support system as well as being used by the estimating staff. This is in spite of the fact that it was not intended to be an evolutionary prototype. The developers paid particular attention to educating the users of the dangers of adopting rapidly built throw-it-away prototypes as fully implemented systems. This has lead to problems that were not anticipated. In particular because the prototyping tool was not chosen with the intention of building a finished system, installation and maintenance were not considered relevant at the time. However it is difficult to deny access to a prototype even though it was never intended for everyday use.

Overall this part of the work was considered to be successful. The change control strategy was useful, particularly as it imposed a discipline on developers to record and report changes in a systematic manner.

7.7 Final prototyping work carried out

Two additional exercises were carried out which form the final part of the commercial action research work at the company.

7.7.1 Precious metals pricing system

A system for pricing precious metals was designed and implemented in approximately a fortnight. It was built using interface prototyping as the main functions were available in a clearly laid out set of procedures. After approximately a year in use it was significantly enhanced to include additional finishes that had been brought into production. This required additional interfaces to be built to access further data. The system was very easy to write and install because of certain characteristics:

- It was small in scale
- It was designed for the use of one person only
- The processes required were available in a well defined form
- The user involved was familiar with prototyping and co-operated well in a positive fashion.
- The developer was familiar with the software used to build the system
- The developer was used to using prototyping and was able to respond to criticism positively.

Only 2 sessions were conducted, the first to establish the main interface characteristics required and the general output, the second was to clarify the quickest way to pick up data from tables presented to the user.

It could be argued that prototyping was not the most effective way to build this particular application as the requirements were well understood. However even in this example user feedback was required to clarify certain implementation issues which without doubt led to an agreed implementation in a very short time scale.

7.7.2 Main route and pricing system

After the silver route system had been implemented it was decided to extend the system to process silver costing as well as determining the route through the factory. At the same time other processes were to be included in the system. In the first instance the nickel plating shop was chosen as the next area of investigation.

This part of the project involved 4 developers and 2 main users (although others were also involved on the periphery). Over 15 prototyping sessions were carried out and the end result was a large and complex rule based system with an interface to over 12 external data files. The project lasted approximately 1 year in duration.

Ultimately this system was a failure and although tried out on several occasions was never fully implemented.

It suffered from the following major problems:

- The boundaries of the system were not clearly defined by the client organisation.
- It was very difficult to test the system against existing data and achieve consistent and meaningful results
- The system attempted to build in costing information based on marginal job shop costing which had never been tried before in the company.
- The system was very unwieldy and attempted to carry out computation based on data which was of dubious reliability
- The development team attempted to build too much into one system.
- The use of case based technology should have been investigated as an alternative form of technology for this problem. [1] (Despite being in its infancy at the time of the investigation)

Given the success of the other systems that had been built and the experiences of the team by the time that this system was underway it is clear (although with the benefit of hindsight!) that the project should have been approached in a different fashion. In

[1] Stottler (1992) is a good example of how Case based technology was applied to a similar problem.

particular the project would have benefited from a time box type control mechanism which would have highlighted the major problems that were likely to occur earlier on in the project.

7.8 Results of projects carried out

The major points of the 4 systems built are shown as follows:

	System 1	System 2	System 3	System 4
Purpose of System	Standard Information System	Silver Shop Routing System	Precious Metal Pricing System	Complex Metal Pricing System
Prototyping Type	Interface	Functional	Interface	Interface and Functional
Delivery	Incremental	Incremental	Evolutionary	Monolithic
Number of Iterations	Not Recorded (But>3)	6	2	Approx 20
Language	RPG 2 on IBM 36	Crystal on IBM PC	Crystal on IBM PC	Crystal + Lotus 123
Purpose	Implementation	Specification Clarification	Implementation	Implementation (Specification as well)
Size	Medium	Small/Medium	Small	Large
Number of Users	3	2	1	4
Number of Developers	2	2	1	4
Control Mechanisms	Light - Meetings only, no change mechanism	Change Control + Review meetings	None	Change Control + Review Meetings
Success?	High	High	Very High	Failure

Table 7.2. Main attributes of systems built at the company

7.9 Results and Conclusions

It would be fair to say that prototyping was used with a good degree of success to build systems that were enthusiastically received and performed well in the tasks that they were designed for at the company.

However the failure of the final (and largest) system illustrates that as projects become larger there is a need for control mechanisms to be planned and correctly enforced. As mentioned above, it seems reasonable to conclude that although the System 4 project appeared to be running well on a week to week basis, it was suffering from a lack of project control and direction at a higher level. I.e. the mechanisms that reviewed progress towards the final goal were not correctly implemented. As progress was steadily being made it was assumed that the natural progression would be a large and successful system, when in reality the final outcome was a large and unsuccessful

system! Mechanisms to prevent this problem must be in place, particularly for larger projects where the risk and consequences of failure are greater.

Hence with regard to the original aims and objectives of this part of the project, the following points were noted:

- Keep extensive written records of each iteration (and prototyping sessions) to ascertain how much this could help improve the project control;

Documenting the details of how one iteration of a particular piece of software developed in to another turned out to be a time consuming job. However as an archive of work carried out, it did prove to be useful when the reasons behind particular decisions needed to be checked. On the basis of the work carried out as part of this study keeping the project documentation accurate and up to date should be encouraged as part of any project using prototyping.

However the ultimate failure of the System 4 software suggests that merely logging historical evidence from the project does not in itself help overall project control. It might be possible to review this information against an original project plan in order to improve project management control, but from the work carried out here it would appear that further higher level project management controls are required.

- Use more than one developer at prototyping sessions to see what effect this had on the sessions carried out;

The effect of using more than one developer at prototyping sessions was generally beneficial as results from sessions could be cross checked. Theoretically an equally (or maybe even more) accurate record of events could be maintained by using tape recordings or video. However this type of approach is not always acceptable, and additionally some individuals are likely to be more inhibited even if this form of recording is used. A further disadvantage is the time required to transcribe the most important pieces of information from other superfluous material. Hence the use of two or even three developers at prototyping sessions is likely to enhance the value of the information obtained.

- Use of a change control strategy to try and improve overall project management.

The use of the change control strategy was certainly useful (particularly in System 1). It was beneficial to see how many changes were required and how serious they were in the various sessions that took place. By assessing the complexity of the changes required it was possible to give the client useful feedback on the impact of their requests on the software.

Unfortunately the use of this strategy did not prevent the failure of the System 4 development. It should be stated that this was probably not the fault of the technique, but due to a variety of other factors which were not addressed (See section 7.7.2). This piece of research further highlighted the problems of management and control that were also documented in Chapter 6. As a result of the action research being carried out it became apparent that projects involving prototyping must have adequate high level project management built in to them at the outset in order to recognise the symptoms of failure that manifested themselves in the System 4 development.

Chapter 8. Prototyping and the use of function points

8.1 Aims and objectives

The earlier experiences of prototyping development illustrated that there seems to be a particular need to control prototyping at a higher level than on a short term (typically weekly) basis. Unless this is carried out it is possible that overall (high level) project progress will be ignored. The following chapter looks at the application of function point analysis applied as an aid to helping the estimation, monitoring and control of the prototyping process.

Hence the main aim of this section is to investigate whether the use of function points can help in the estimation, monitoring and control of software development projects using prototyping. Although important to the commercial side of the project it was not the intention of this exercise to assess prototyping for its use as a communication vehicle or general systems development aid, this was part of the previous exercise.

This part of the investigation was carried out as an exercise in action research where the work carried out in the local authority by the developer was directly influenced by their presence.

8.2 Background

This chapter details the work carried out on the development of a small system for a district council. The software required was constructed by a single developer from a draft specification prepared by a district council. The requirement was for a system to help with the control of building and repair work for the housing stock belonging to a local council.

The system was developed using prototyping and incremental delivery, where completed sub-systems were delivered in stages. The work also involved the functionality of the system being assessed using Mark 2 Function Point (FP) analysis. Some tentative conclusions are drawn regarding the combination of FP analysis and incremental delivery (see Chapter 1). This method of estimation was chosen as there is evidence (Symons 1992) that it is currently the most effective quantitative method for estimation and productivity measurement. The so called Mk 2 method is an enhancement of the original Albrecht work on FPs (Albrecht 1987). Whilst it retains the advantage of being independent of lines of code it is also compatible with the use

of Entity Relationship (ER) diagrams. The work reported here used these in the system design so it was felt that this technique would be the most appropriate.

The main developer chose to use prototyping as the method of developing the specification from which the sub-system designs were derived. By combining the use of prototyping and incremental delivery it was hoped to deliver completed sections of the system which achieved the following aims:

- Maintained high user interest and involvement throughout the project
- Delivered sub-systems which accurately reflected user needs
- Allowed the system to grow in stages thus giving users parts of the system which could be used prior to entire system delivery

During the system development process a log was kept of the time spent on the different types of activity that were carried out; these categories are listed in the analysis section. The intention of the project was to try and derive a relationship between FPs counted at the onset of the increment development and the time to develop and deliver the increment. In addition the functionality of the final delivered increment was also assessed using FP counts. Hence it was hoped that the control of incremental delivery using pre and post measure of functionality could be investigated.

There were certain points which required particular consideration during the exercise:

- The use of prototyping would actively encourage iteration and change. Hence the original specification and system design would require revision
- Function points counted at the start of the increment would not represent the final functionality of the delivered increment (reflecting the point made above)
- The control mechanisms for determining which changes should be implemented and which should not, had not been formalised at the onset of the development
- Changes to a specification and subsequent design may result in reduced functionality, but work would still be required to achieve this. Hence resource

requirement is likely to be a function of the sum of the modulii of the changes in the increment rather than a simple positive relationship

8.3 Productiveness

The final point listed above raises a very important issue regarding productivity. Changes required which reduce the required functionality of an increment will have the effect of apparently lowering productivity (in terms of time and resources used versus delivered functionality). It could be argued that these features should never have been introduced and are the result of over enthusiastic system developers! Nonetheless it has been suggested (Mayhew 1992) that the term productiveness is used to assess the value of a system where this is a measurement of value to the business rather than a simple effort and cost figure. As a result the effectiveness of the prototyping process should be based on this metric plus control using time-boxing (Graham 1991). Time-boxing (as the term implies) suggests that a development is split up into sections based on pre-agreed time units. It is recommended that a typical time-box should be of the order of 6 to 10 weeks. The development is reviewed after this period and a decision on the next stage made. The reason for carrying out the work this way is to prevent developments degenerating into uncontrolled and lengthy exercises striving for unachievable functional goals and creating serious problems of late or non delivery of projects.

8.4 Work carried out

The first part of the project involved the delivery of four increments. The intention was to use these as a pilot calibration exercise for the entire project. For these four parts the functionality of the system was measured **after** each increment had been delivered. There was no analysis of the initial system design with regard to functionality, and therefore the changes implemented as a result of prototyping were not analysed and hence neither the productivity or productiveness of the prototyping process could be measured. However a thorough analysis of all activities during development was kept and this was analysed in order that the process of prototyping could be evaluated during different increments.

It was the intention to do this for the second stage of the project when a further set of increments were to be built. Unfortunately this was not possible.

It should be noted that the work involved in the project was carried out by 3 people, their activities were split as follows:

i) Main system development and activity logging.

ii) Function points counted for each increment. (This also involved the system design being re-drawn to include full entity relationship (ER) diagrams required because of the use of Mk 2 FPs).

iii) Derivation of relationships between time, activities and function points.

8.5 Work Classification Key

In order to classify the development work, the activities carried out were divided according to the following criteria (these codes are also used on the graphs listed later in the report):

Activity Code		Activity
MOD	-	Module hierarchy production/amendment
INIT	-	Initial sub-system prototype production
CHNGS	-	Implementation of changes requested
PRA	-	Prototype process administration
PR	-	Prototyping session
PILOT	-	Conversion of prototype to pilot system
TEST	-	Final testing of increment
CONV	-	Conversion of basic system to production quality.

Each increment within the system has a unique coded identifier, these are listed below. The codes of each are not significant:

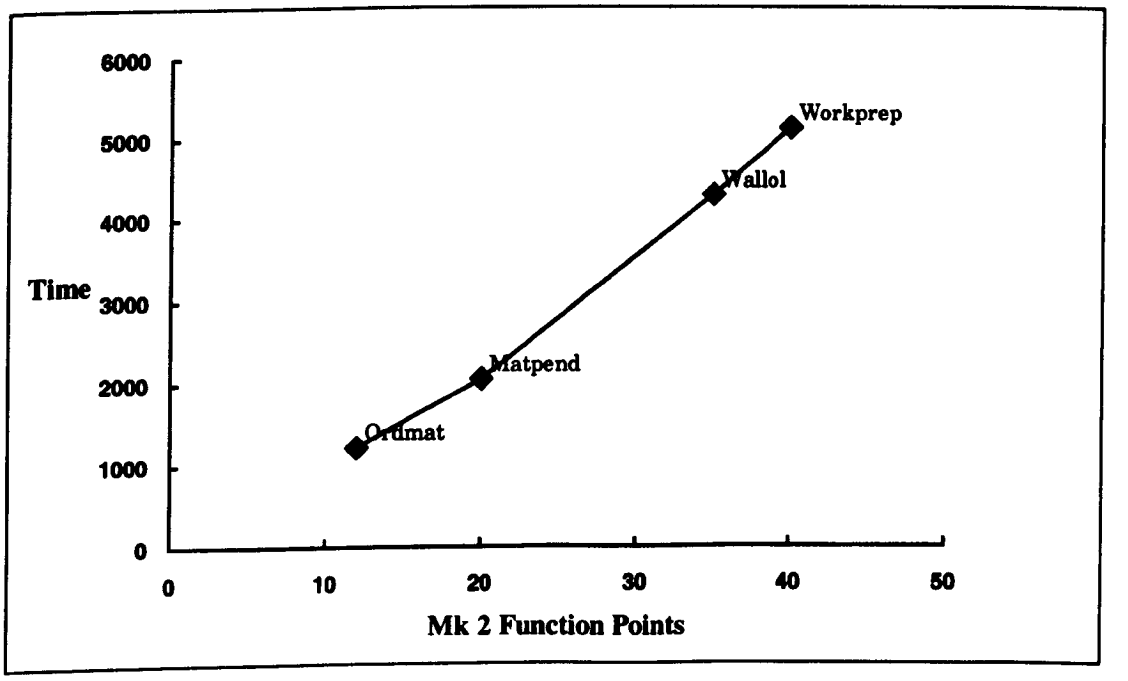
Increment Code		Increment
WORKPREP	-	Work preparation for contractors
ORDMAT	-	Order of materials
MATPEND	-	Materials pending
WALLOL	-	Work allocation

8.6 Results

8.6.1 Function points and development effort

Graph 8.1 shows the relationship between the overall functionality of each increment and the time taken to produce it. From these sub-sections it appears that there is a good correlation between function points counted and overall development effort. The relationship, whilst not completely linear suggests a strong association between the two factors. However with only four data items it would be unwise to draw definite conclusions from this. More data is required to substantiate the correlation. Nonetheless, the overall trend does point to a good relationship between time spent and FPs counted.

It should be noted that the FPs counted are **after** all modifications as a result of prototyping have taken place. For this part of the project no count based on initial analysis was carried out. Unfortunately this means that a comparison of specified business requirement requested (in terms of function points) and actual delivered system (again expressed as function points) is not available. Hence the use of function points as a **predictor** of development effort could not be assessed in this exercise.



Graph 8.1. Time taken to deliver one function point (in minutes)

8.6.2 Estimation

In order to be able to use this form of development it must be shown that accurate estimation techniques can be applied.

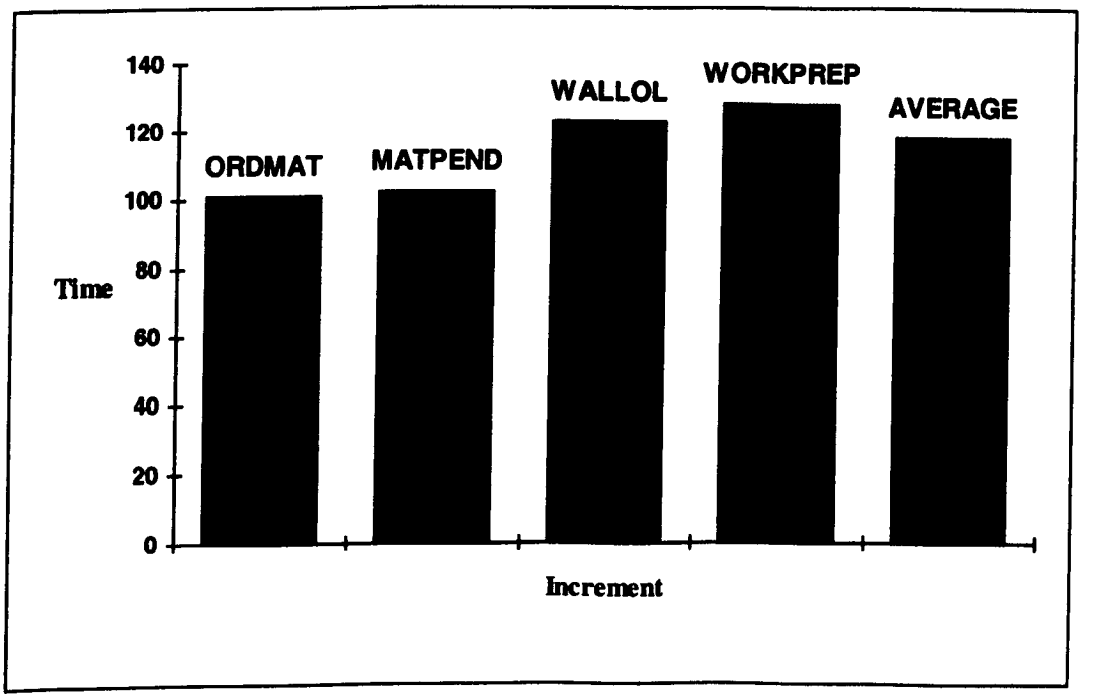
There are various ways that this can be achieved:

- A project can be compared against previous projects
- A project can be sub-divided and later sections calibrated against earlier work carried out.

The two methods can also be combined. In the project described here, it was difficult to make a good estimate using previous data for a number of reasons:

- The application area was not familiar to the developer
- The developer was not familiar with the operating system and development environment (DATA/BASIC running under PICK)

Hence previous analogous data was not available and therefore it was important that a calibration exercise was carried out.



Graph 8.2. Average time taken (in minutes) to implement a function point for each sub-system

From the data supplied the overall level of function point delivery is approximately 1 FP per 119 minutes of effort. The variation in FP delivery rates is 101, 102, 123, 128 minutes per FP (see Graph 8.2). 119 represents a weighted average delivery rate. The values for effort per delivered FP increase with increased development size. This is broadly in line with most published work on productivity and project size. The absolute values of functionality should be compared directly with other published FP work with caution. The data here is likely to overstate the productivity achievable as final user documentation was not produced as part of each increment.

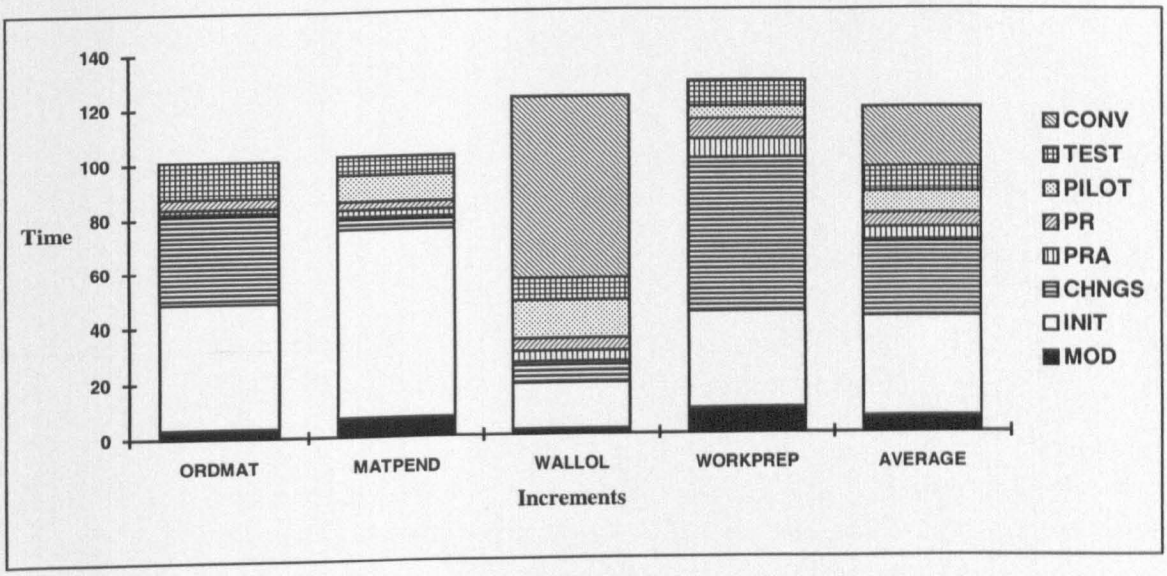
8.6.3 Sub-division of effort within increments

Graph 8.3 shows how time was spent on each sub-section of the system. The graph is scaled so that the total development time is divided by FPs delivered. The total column height represents productivity, I.e. Effort required per delivered function point. The final column represents the average over the entire development.

From this it can be seen that in 3 increments (but not WALLOL) the bulk of the development effort is spent carrying out the initial sub-system prototype production and (in 2 cases from the 3) implementing changes suggested at prototyping meetings.

In each case the initial module hierarchy production and the prototyping sessions and prototyping administration did not form a major resource overhead. In one case production of a pilot from an earlier prototype involved greater work than implementing changes requested by the user group.

This raises a problem for monitoring and estimation. The data suggests that the initial preparation work is likely to be at least 30% of the overall development effort. However it can rise to around 70% of the total effort. It is possible that the use of Mk 2 FPs in conjunction with ER diagrams could enable estimation to be done very early on in the development process. However the accuracy of the estimates produced will be heavily influenced by the level of iteration and change that occurs as work progresses. Further work on this would be required to assess its value.



Graph 8.3. Breakdown of time taken within each increment

In graph 8.3 it can be seen that the figures for the WALLOL and MATPEND developments are substantially different.

For WALLOL although the overall productivity rate are in line with the other parts of the system, a large part of the total figure is taken up by the conversion of the software into a standard suitable for incremental development.

On investigation it was discovered that the increment involved a high level of throw-away prototyping. As a result a significant effort was required to reproduce the specification required in the form of a system of suitable quality. It is interesting to note that despite this radically different approach overall productiveness remained in line with other increments.

A similar phenomena occurred with the MATPEND increment. In this part of the development the largest amount of time was spent developing the initial prototype system. Despite this, the overall productivity within this particular increment remained much in line with other parts of the system.

In order to be able to monitor the productivity during a development it is necessary to discover if there are any indicators that can be used to predict the effort likely in a very early stage of the project. From the work carried out in this project there is certainly some evidence that incremental system delivery using prototyping can be estimated in advance. It may even be possible to estimate the likely total development time once some benchmarks have been established, but the efforts on particular parts

of the development do not seem to conform to a pattern. Further analysis is required in order to analyse the relationships - if they indeed exist.

The monitoring of project progression may be possible by evaluating function delivery, however there are certain points that need to be considered:

- From this work it appears that Mk 2 FPs could be used to give guidance on the duration's of the time boxes required **within** an entire project.

- Within each increment this work indicates there may be some difficulty monitoring progress on a micro-level (i.e. intra-increment) using an FP approach

However if the time box duration's are estimated using Mk 2 FPs, then progress and reviews could be triggered at sensible points during the development. The work carried out showing the relationship between function point delivery and effort supports this.

8.7 Discussion

From the data supplied it is possible to make some general predictions of future efforts for other parts of this system which could be generalised for similar projects in comparable environments.

1) The total development effort per delivered FP should take about 2 hours. The figure should be decreased for small systems and increased for large. (This productivity figure is artificially high compared to systems delivered with full system and user documentation).

2) Within each development sub-section, initial prototype development and change implementation should take the bulk of the development time (of the order of 80%). However the WALLOL increment presents an anomaly to this relationship - hence the nature of the problem under development is likely to heavily influence the accuracy of this metric.

3) With the data available it is not possible to give an estimate of the likely split between the time taken for initial prototype development and change implementation. This is a serious drawback as unless it is possible to derive a reasonable relationship

between the two across an application, the estimation and control processes will become very difficult to manage.

4) Conversion from final (evolutionary) prototype to operational system is not a large overhead and should take less than 10% of the total development effort. It is impossible to compare this figure with other studies as to the author's knowledge no comparable work exists.

5) It is possible that an estimate based on an ER diagram is likely to be valuable early on in the development. This figure should be enhanced with data from prototyping sessions as the level of iteration becomes better established.

6) The possibility of a combination of function point analysis (for entire increments) and time boxing (for control within increments) could be adopted for project control. This means that although estimation is possible, low level monitoring will be hard to carry out using delivered functionality as the measure. Hence a combination of time spent and functionality delivered is recommended. More work is required to validate this assertion.

7) Time-box lengths can be derived from function point analysis where possible, but it is recognised that this might not always be practical, in which case alternative methods of generating time-boxes should be investigated.

8) The relationship between requested and delivered functionality was not possible to derive in this study and remains an important area for further research. The author is not optimistic that such relationships can be easily derived and expressed in straightforward empirical terms.

With regard to the particular aims of this part of the research project the work carried out suggests that a development using incremental prototyping can utilise FP analysis as an aid to estimation, particularly after one or more increments have been completed. However using FPs as the sole form of estimation mechanism is unlikely to be effective, particularly at a micro level where the time spent on different activities can vary widely. From these results the monitoring of productivity within an increment does not seem to be feasible using a FP based technique and hence cannot be recommended.

8.8 Discussion on the use of action research

It should be stated that the results obtained here can only be directly applied to the particular studies at the time they were undertaken. It is also apparent that it is impossible to control the environmental factors that are in place at a particular time, hence the personnel involved, their attitude towards the investigation, the personal views of the researcher, the operating conditions both within and external to the organisation plus other factors cannot be replicated in laboratory type conditions to allow a duplicate exercise to take place.

However it is the view of the researcher that this type of study can give useful material on real problems which can be of use to the organisation itself and also (if published) to the wider community at large. Hence despite shortcomings (which must be acknowledged), this type of research work can provide useful research material.

Part 3. Model Design, Experiment and Conclusions

The final section of this thesis is split in to three components. The first section details how a model for incremental and evolutionary development incorporating prototyping and RAD has been developed.

The second section outlines an experimental field study set up to examine various aspects of RAD and its effect on software quality and software project management.

The final section concludes and summarises the research work carried out. It evaluates the use of the particular approaches that have been taken as part of the project. It examines whether the aims and objectives have been fully met and additionally considers where further research could be undertaken in this area.

Chapter 9. A model for control - the total prototyping methodology (TPM)

The review of existing work in the area of prototyping illustrates that controlling the progress of a RAD or prototyping project, particularly if it involves several people and is of significant duration (e.g. greater than 3 months) does need careful planning and monitoring. Although the case study evidence (see part 2) illustrates that prototyping or highly user centred development does have some positive affects on the development of systems, there still exists a need for effective control and organisation of the process. This chapter sets out a model that can be used as a framework for software development using a RAD or prototyping centred approach.

9.1 The importance of controlling the process of prototyping

By identifying control as a particular difficulty associated with prototyping, it is evident that the main problem is one of trying to control a **process**. I.e. what is actually required is a means of initiating, continuing and concluding that process so that the development is carried out and concluded with the most efficient use of available resources.

This point has been considered by several researchers in the field. E.g. Lichter, Schneider-Hufschimdt, Zullighoven (1993) state:

> "control techniques developed for traditional life-cycle oriented software development tend to hinder software projects using prototyping approaches."

When using prototyping this inevitably means controlling a process that encourages iteration. The idea that the creation of information systems should be regarded from a process point of view is quoted by Checkland and Scholes (1990). It is suggested that the "project life cycle" approach to a computer project will only apply to special cases where the task is not contentious or is mechanistic, and that in other areas this method is not suitable. This backs up the view of the author that the problem of control must then be examined from a process perspective.

9.2 The basic stages of the prototyping process

If this idea is extended into the basic prototyping process it is suggested that there should be four basic stages. Naumann and Jenkins (1982) suggest the stages are:

- Requirements identification
- Prototype development
- Implementation and use
- Revision and enhancement.

Mayhew *et al* (1990) describe these as:

- Deliberation
- Preparation
- Organisation,
- Actual prototyping and completion.

This type of approach is also very similar to the Evolutionary Delivery Model suggested by Gilb (1988). This is shown in figure 9.1.

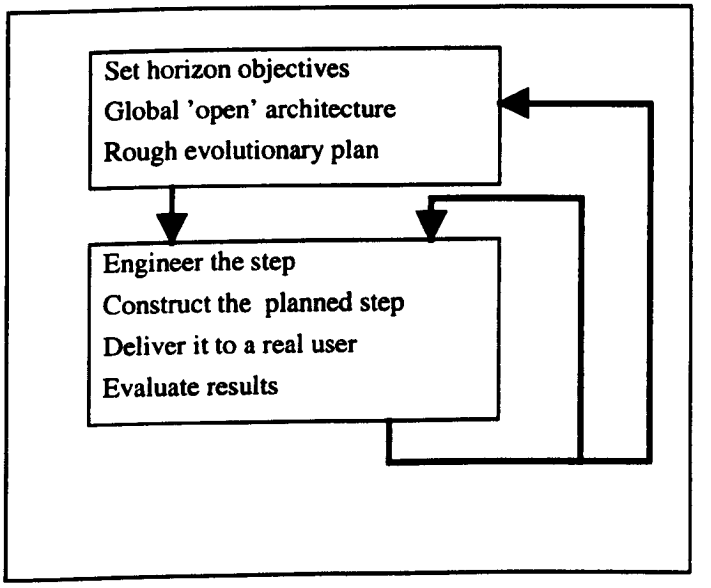


Figure 9.1. The Gilb evolutionary delivery model (From Gilb 1988)

In the smaller loop it can be seen that each iteration has four steps associated with it. In some respects this model and the others mentioned do have similarities to the waterfall model. The main difference is that change is encouraged and expected which

means feedback and iteration are essential in the various prototyping and evolutionary development models.

If these models are considered from a process point of view it is evident that in order to ensure that the process is optimised and planned methods are required to:

- Estimate how long the development process will take. (Estimate Model-EM)
- Monitor the progress of the project against the estimate. (Progress Model-PM)
- Determine the point at which the process should finish. (Finish Model - FM)
- Evaluate and document the progress to date. (Evaluation Model - VM)

Therefore a complete or total prototyping model (TPM) comprises these four sub-components. This is illustrated in figure 9.2.

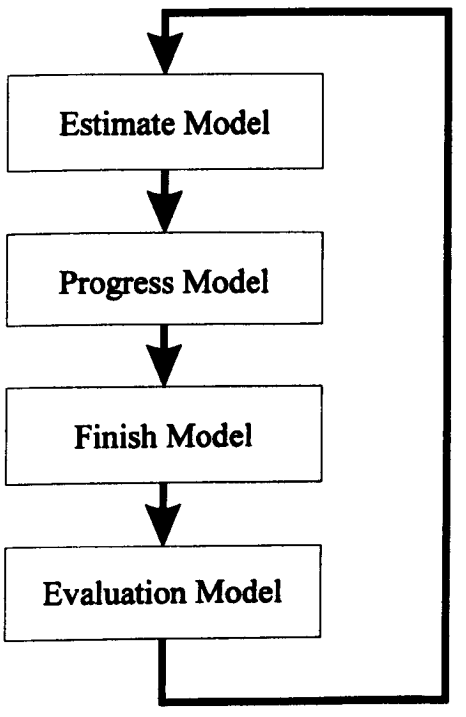


Figure 9.2. The total prototyping model (TPM) with basic sub-parts

It should be noted that the structure is similar to all of the models cited. This is intentional. The idea behind this work is not to suggest a new method for prototyping development, but to suggest how prototyping as it exists can be structured and

controlled. The four stages are fundamental parts of the iteration loop. This idea was first considered in Stephens and Bates (1993).

The emphasis on controlling the process is the most important part of the TPM. Madhavji (1991) recognises that one of the fundamental flaws of methods based on the life-cycle model is lack of control over the way a project proceeds. He comments on the use of life-cycle models that:

> "A limitation of these models, however, is that they hide important process details that are crucial for the success of software projects."

In the view of the author the lack of weight applied to such processes is a seriously overlooked part of published software design and project management approaches. Such process details include:

- How an activity is triggered or terminated
- The role of humans and communication between them in the process
- The identification of parallel and sequential process steps
- The notations, methods and other tools that could be used in different steps

Madhavji concludes that although some of these points may be represented in documents, the development work cannot evolve as the needs of the project change.

This idea is also explored in Veraart and Wright (1995) who state:

> "The concept of a software engineering *process* is of an identifiable entity that can be modelled as a set of identifiable activities that can be described in terms of its tasks, inputs and outputs, entry criteria and exit criteria and that can be controlled, measured and adapted to suit the needs of both the organisation in which it is performed and the current development project on which it is performed."

The TPM approach addresses this by considering the methods for *controlling* the development, not by prescribing the techniques that should be used in carrying out the development itself.

The nature of software development is such that the components identified within most development methods would be the main constituent parts of the TPM. I.e. Some combination of analysis, specification, design, testing, documentation, and implementation. However the overly prescriptive nature of typical development methods can be avoided by assigning each part *as appropriate* to a TPM instantiation. Hence, for project documentation; time and effort would be estimated, a suitable development method and means of measuring its progress would be agreed, and a finishing procedure would be fixed and finally the documentation would be produced in the specified form. I.e. the sub-part of the system (documentation) would itself be carried out in accordance with the steps of the TPM. It is logical that any component of this model should be approached using the steps outlined. This may seem over-elaborate, but it is necessary to maintain consistency. The detail required for the estimation and monitoring of each task will obviously vary according to the envisaged size and complexity associated with it. As a guideline it would seem appropriate that a task considered to have any uncertainty associated with it should certainly be judged to be a potential problem (unless very trivial) and be tackled in a comparatively formal way.

As this approach is aimed at control. It is not good enough merely to say that "documentation will be produced". Particular techniques that can be used to formerly control the process should be identified in order to enforce control.

In order for the model to be usable, each component should be self-contained. This is necessary as developers may (for example) already have a proven estimating system, but require a monitoring and completion model in addition. Hence each part of the TPM should be adaptable to the particular conditions which are prevalent for a given development. The method must allow existing expertise to be utilised in order for it to be useful. E.g. If an organisation already uses a function point derived estimation model successfully to estimate project cost, then this should be used as part of the TPM model and form the bulk of the EM part of the process. In these circumstances it would be foolish to prescribe another method.

9.2.1 Estimation and risk (EM)

The main difficulty of trying to set estimates is that it is very difficult to plan for unknown change and therefore the risks associated with it in the future. This is not a problem which is unique to prototyping. There are numerous examples in the literature that reflect on this problem. E.g. Redmill (1989) states:

> "Without an understanding of the full range of the users' requirements, it is not possible to estimate the project's resources, cost, and time."

Unfortunately developers cannot say to potential clients that a time and cost for a development cannot be estimated. The nature of commerce is such that this reply is not likely to be accepted! However since evolutionary development embraces change as an inevitable outcome of the software development process, the estimating model designed for such development must recognise that change will occur. It may be possible to link the likelihood of change and hence the elasticity of the estimate to some other parameter, (E.g. To the comprehensiveness of the users original specification), but little practical work is available to help with this problem.

Additionally it is important to note that software developers are becoming increasingly aware of the difficulties of estimating in general. More development methods are linking the software development process to estimates of risk as well as cost or time (E.g. Boehm 1986 is a commonly cited example). Hence it makes sense for any estimation part of a software development model to take account of this and have some form of risk assessment built in as part of the estimation process. The TPM requirement is that a formal assessment using previous data is likely to produce the most accurate estimate. The use of analogy (E.g. Cowderoy and Jenkins 1991, Shepperd, Schofield and Kitchenham 1996) or a comparable approach using past data and a commercial estimating tool is likely to provide a good foundation for systematic estimating. If organisations do not have past data then other techniques such as the wide-band Delphi approach could be utilised. Data from such estimates should be recorded in order that a historical base can be built up to improve the process. As the base of experience is recorded, it should be possible to amend and modify estimation techniques to produce more accurate predictions of effort, risk, time, and resources required.

As with each part of the TPM, it is not good enough to merely give "an estimate". If the eventual aim is to try and provide a formal set of guidelines, then decisions or data that influence the decision must be recorded and classified in a meaningful way. This point is emphasised by Londeix (1987) who states that to be useful, estimation "must be based on an estimating method".

It should be observed that if any of the main points of the model are ignored then the global aim of controlled software development is likely to be lost.

9.2.2 Progress monitoring (PM)

Monitoring the progress of development has traditionally been one of the most challenging aspects of software development. Today evidence from the computing press (Robertson 1996) suggests that this is an area that is still causing difficulty. However, only by effective monitoring once a project has been initiated is it possible to ensure that deadlines are kept.

There are two basic approaches to ensure that development does not become an infinitely long process. It is necessary either to specify a specific set time for that part of the development, or alternatively to set a target, related to a pre-specified level of development (in effect a functionality level). In each case it should be noted that development is planned according to a stated goal, either in terms of time, or in terms of a set of agreed deliverables. Examples of each method are set out below.

The time-box approach

Graham (1991) outlines the way such a method is used as part of the BIS applied systems rapid development method (RDM). The main control element of this method is to set a rigid limit to the time available for part of the system development. The experiment that is outlined in Chapter 11 uses this form of control in an attempt to keep the development on track. This will typically be of the order of 40-80 days. By adopting this strategy Graham suggests that certain management issues can be tackled.

These are as follows:

- Wants versus needs - By forcing requirements to be prioritised by negotiation between users and developers
- Creeping functionality - In the traditional life cycle the long delay between specification and delivery can lead to users requesting extra features. The use of a time limit reduces this tendency
- Project team motivation - The developers can see a tangible result of their efforts emerging
- The involvement of users at every stage reduces implementation shock

This form of control does ensure that developers and clients maintain close liaison over the progress of a project. However it still requires good management to ensure that each time-slice is achieving progress. TPM could be combined with a time-constraining approach such as this. Each time box could contain one or more iterations around the prototyping loop. Hence each stage from estimation to evaluation and documentation would be contained within the nominated time box. The same principle can also be applied in a similar way to cost.

The target related approach

The controlling factor behind this form of management is to have a target set in terms of system performance or functionality. In order for this to be achieved it is necessary for the progress of the development to be measurable in some way. A method to do this is exemplified by the GQM technique (Basili 1985). The principle behind this form of control is to define project goals (tailored to a particular need). These can be refined into a set of quantifiable questions which can then have a specific set of metrics for data collection. This form of progress monitoring again fits in with the TPM approach in that a time or effort estimate can be constructed and the progress towards it measured using a defined metric.

If an estimate based on an agreed level of functionality is set up then it could be expressed in terms of a number of function points (e.g. Low and Jeffery 1990). The progress of the project can then be measured as the progress of the development towards the agreed level of functionality, expressed as the number of function points delivered. If the progress made drops below a certain level, then a goal revision or reassessment of that particular part of the development should be undertaken. A minimum acceptable level of progress could be agreed between developers and users at the time of the estimate. Adopting this tactic must be carefully considered as there is likely to be as much responsibility with the user/client organisation for progress to be made. In this context the term functionality should take the widest possible meaning. The "functionality" could be defined in terms of performance, or "completeness". The main issue being that whatever indicator is chosen, it should be expressed in a way that can be measured. However evidence from the research (see Chapter 9) suggests that this form of measurement may well have drawbacks for prototyping and RAD developments.

This method of measuring "deliverables" will require tuning to the environment and organisation that it takes place within and so is unlikely to achieve good accuracy

when initially adopted. In common with the other techniques discussed it does provide another quantifiable component to help formally control the software development process.

9.2.3 Determining the completion point (FM)

If the project estimation has been set up using a formal technique the point at which the project or particular iteration finishes should be easy to decide! Given the criteria for progress monitoring outlined above, there are a limited number of criteria that dictate that the completion point has been reached. If the estimation has been time based, completion occurs when either of the below has been reached:

- Completion is agreed before the estimate deadline
- The time deadline is reached

If the estimate has been based on a given level of functionality the following will determine completion:

- The agreed level of functionality is obtained
- The progression against a standard is too low, triggering a review

Whether a functional or time-based approach is used it can be seen that in order for completion to formally occur, a completion point must be agreed based on an estimate. If an estimate is not produced, the notion of "completion" has no real meaning.

A failing of methods that dictate what processes need to take place, but do not specify how they can be monitored will always be that the point of completion cannot realistically be stated. If a formal completion method is agreed, the responsibility for its enforcement again lies jointly with both the developers and clients. Both parties should understand the process and both have the means to judge whether or not the project is progressing and ultimately finishing on time. Again the responsibility and importance of the user is of paramount importance in the process.

9.2.4 Evaluation and documentation (VM)

The final stage of the TPM is the evaluation and documentation of the work that is carried out. There are three possible outcomes to this process:

- Stop and do not continue with any further work on the project
- Stop at the end of the current iteration. The system will be implemented either as it stands, or with any necessary rewrites or modifications that are agreed
- Additional work is required and desirable, involving further iteration as is deemed necessary.

It is important to note that in terms of a project the line between the second and third options is not absolute. Using traditional software life-cycle terminology, the end of the development phase is generally marked by the start of the maintenance stage. If evolutionary development "language" is substituted, it may be that a system is delivered and becomes operational, but that a further iteration (maintenance, or enhancement) of the loop is required. At this point it becomes questionable whether the project has finished and a new one is starting or whether it is merely an continuation of the existing exercise. It is impossible to dictate this without the particular circumstances of a project being evaluated.

The final documentation is an integral part of the development. The form of the documentation and the exact role it plays in the project must be understood by both parties. The development ought to have a specification in some form other than just the working system, to facilitate maintenance over its anticipated lifetime (Smith 1991). The nature of that specification should be sufficient so that future developers can use it for *any* further work that is required. The particular form of notation used to represent the system is not primarily for the use of the clients. It should state clearly and unambiguously how each part of the system works and what each part does. The notation (diagrams, natural language, decision tables) used should be chosen so that it offers the best representation to the developers. The developers will also have a copy of the software, which is (or should be!) an animated form of the specification. The client organisation for commercial (contractual) reasons are likely to require a limited form of the specification which should explain what the system does. The method by which it does it is likely to be of little value to them.

In contrast to structured methods the nature of the specification is not dictated by the method chosen. The method does not drive the development, rather the type of development should dictate the method and hence the documentation required.

The final model is represented in figure 9.3. The boxes on the left show possible alternative methods that can be used to "plug in" to the TPM. They suggest various alternative examples that organisations can use to drive and control the overall process.

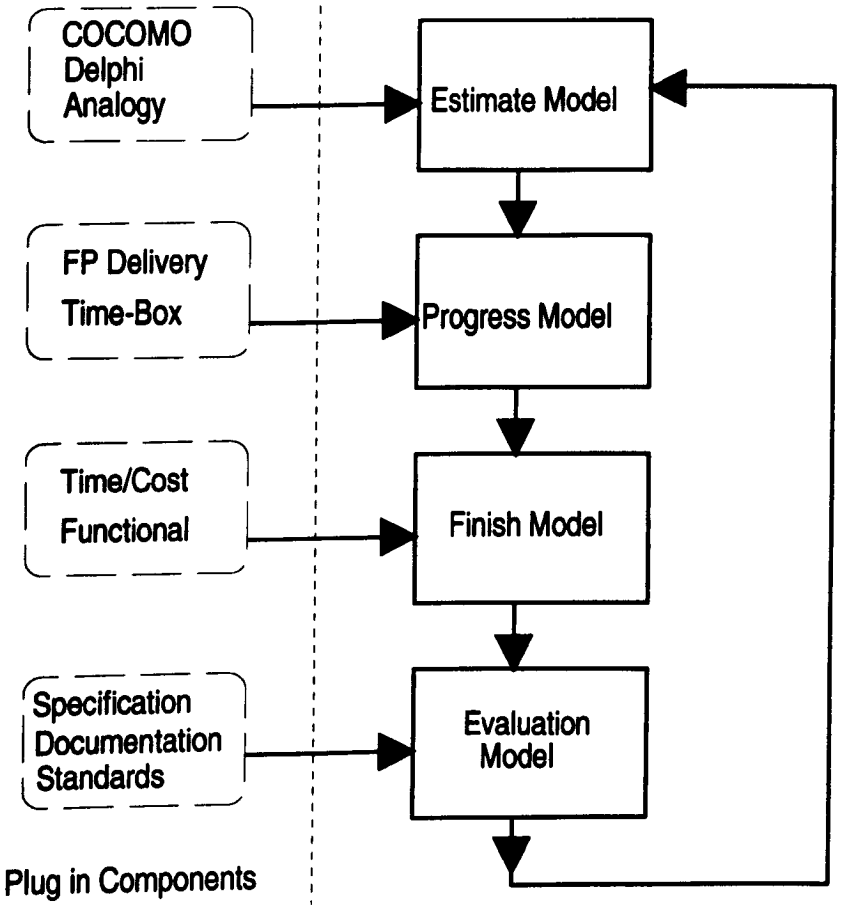


Figure 9.3. The total prototyping model with example sub-components

Unfortunately although it should be possible to construct several different implementations of this model, it must be recognised that most forms of measurement within software development are still very limited. Lehman (1991) notes that:

> "In general, however, measurement technology is still without an underlying theory, is still groping for 'useful metrics'.

Further development of measurement technology and its integration into environment design is of fundamental role and must become a major research focus and goal."

The TPM can only be useful if the formal controlling and measuring methods can be shown to be consistent.

9.3 Discussion

This chapter proposes that a **total** managed approach to prototyping can be applied. The emphasis is not on the particular micro-level techniques that might be employed to develop software, but on the controlling mechanisms that can be used as part of a total method. It is recognised that prototyping does have very useful benefits, but controlling it is an area that is cited by many authors as causing inhibition of its wider use.

The model proposed is designed so that different estimating and controlling methods can be plugged in to suit the requirements of different organisations and different software development environments. By adopting this approach it should be possible to control markedly different projects using different techniques, but adopting the same overall controlling principles.

In summary without using this form of control it is proposed that prototyping cannot be planned or executed in a meaningful way. The model advocates facilities that use appropriate formal planning and measurement methods, hence allowing prototyping to be managed effectively, with well defined and measurable goals.

Chapter 10. A field experiment in systems development

10.1 Introduction

In order to examine the claims made about prototyping it was decided that in addition to the action research and case study work, an experiment into information systems prototyping would be carried out. This would allow more objective analysis of the effectiveness of prototyping to be made.

The aim of the experiment was:

To discover whether information systems prototyping had a positive affect on the overall quality of software produced, compared to software produced without using prototyping.

It was decided that a study along similar lines to the one carried out in 1984 by Boehm, Gray and Seewaldt (BGS) would be a suitable method for examining claims made by other researchers. However extensive modifications to the BGS method would be made to the way that it was set up and conducted.[1]

Certain parameters were fixed (e.g. time scales and group sizes) whilst other decisions (e.g. development methods and styles) were left open to individual teams. However a TPM style of development was adopted in that a rigid time frame was imposed, minimum documentation standards were required and a minimum delivered implementation was also mandatory at the finish.

Certain guidelines would be adopted as a result of work that had been carried out for other parts of the research project.

Compared to the BGS experiment the significant differences in this one are as follows:

- The experiment was carried out not to test the hypothesis that prototyping encouraged higher productivity, but that prototyping could be controlled and that it could produce "better", more maintainable systems.

[1]An experiment involving 2 teams was conducted at Aalborg University in 1990 - This is reported in Mathiassen *et al.* (1995). It was based closely on the work of Boehm *et al.* (1984)

- A much larger sample size was used (twenty two teams instead of seven).

- The decision to use prototyping or not was left to the teams to decide. There was no compulsion to use it.

- The incentive to produce good software was created by the notion of "selling it" and hence gaining marks for an assessment. The BGS experiment created the incentive by assessing the software in terms of maintainability.

- The assessment of software produced was done largely by students. Staff assessment of software was less compared to the BGS method (although some checking was left in).

With regard to other aspects the two exercises were similar.

The cohort of students used were level two undergraduates. They all had one and a half years experience of using Pascal (the implementation language used), in addition all of the group had studied at least one other programming language or environment (normally C or Visual Basic, or both). Their experience of system implementation in groups was limited, although all of them should have produced systems of some complexity in previously taught modules.

10.2 The field experiment

The study involved twenty two teams of four (occasionally three) developers. The teams were selected by the researcher to try and ensure an even spread of ability across the groups. This was done by ensuring that each group of students had a similar profile in terms of previous grades. Although grades from previous modules studied were used, it is recognised that this is not an entirely infallible method of ensuring teams of exactly equal ability. However it was decided that it was very important to try and ensure that teams were of equal ability to try and minimise the effect that this might have on the various tasks to be undertaken.

Each team was expected to develop their own version of a software system over a seven week period. This represented a typical timebox period that a commercial developer could be expected to conform to and is in line with the way that a development could be set up using a TPM style of development. At the end of the seven week period each team attempts to "sell" their product to as many other teams

as possible at a software fair. Twenty percent of the total marks for a team are derived from the number of sales that are made, hence there is a strong incentive to do this effectively. Every team buys two products at the fair, evaluates them for a week and then uses one product as the basis for a further part of the module. Teams were told that extensive rewriting of the system they purchased and subsequently used would lose them project management marks. The teams were not told exactly what features were required to be added after the software fair, this was considered to be increment or phase two of the development and was designed to simulate as closely as possible the iterative/evolutionary nature of commercial software development.

The main criteria that the teams were told to consider when purchasing software were as follows (these are not in order of priority):

- Conformance to the specification
- Ease of maintainability
- Standards of construction

However it is worth noting that aspects such as interface complexity, use of object oriented techniques and a variety of other criteria are also considered by teams when making their choices.

10.3 The system required

The teams were expected to produce a design (using a CASE tool), a system, test data, and a brief user guide. The system they were asked to produce was a project management tool which allowed managers to view data about the processes that had been undertaken in a project, the personnel involved, tool usage and various other items. The requirements specification supplied was intentionally slightly ambiguous and certainly incomplete[2]. Teams were told of this and were expected to ensure that their product met the users needs correctly.

The teams were told that it was essential that a working product was available seven weeks after the start of the project. They were told that a completely finished product was not required, but that it should be of some value to a user at that point. No time over runs were allowed. This was done to reflect typical commercial pressures and to

[2] The requirements document is included in Appendix 2A.

impose a TPM style of project management. The initial part of the development was considered to be increment one. Development after the software fair was considered to be increment two.

During the seven week period course tutors advised on the planning and project management aspects of the team work. Teams were encouraged to contact the module leader for queries to do with the specification, using the telephone or the University Email system, or through prototyping sessions. All queries were logged.

10.4 Method of evaluating the software

After the software fair when teams had "purchased" software for the second part of the assessment process they were required to fill in test sheets about the purchased software. These sheets were constructed to discover information about five aspects of the software under review.

- How closely did the system meet the specification?
- How reliable was the software in operation?
- How well constructed was the system?
- How maintainable was the system?
- How complete was the system?

The test sheets were part of the assessment process, hence students again had a good incentive to fill them in!

In line with the constraints imposed by the students timetable it was decided that each test form could be filled in completely in approximately an hour. Students were free (indeed encouraged) to test the software further as required by the course, but this was not included in the research work.

It was decided that systems would be assessed by students rather than staff. This was considered desirable for several reasons.

Conformance to specification

The conformance of the software to the specification could be assessed by asking the students to answer some straightforward questions that tested a sample of the properties of the software.

Reliability

The reliability of the system would be better answered by the students as they were required to use it for their second assignment. Hence its behaviour would influence their decision as to whether to use it for the second increment. It was in their own interests to choose a system that performed reliably.

Although the answers to this question are dependent on the individual testers, it was felt that the sample size would compensate for this. In addition testing done by staff would not appear to have any obvious advantages.

Completeness

This aspect of the software could be assessed in a similar fashion to the conformance issues.

Maintainability

It was felt that the students had a vested interest in choosing a product that was maintainable, hence the reasons that apply to assessing reliability also apply to this.

Use of testing sheets

Each member of every team was asked to fill in a test form for three of the systems. The three systems comprised the two systems that had been chosen at the software fair, plus a system that had not achieved any sales. By doing this it was possible to ensure that each system was assessed by at least three different people in the class,

In total approximately two hundred and fifty separate testing sheets were filled in with testing details of the systems under review. As a by product of this approach the systems that had sold more software at the software fair had more test sheets filled in for them.

Additional questions

There were 2 preliminary questions also asked but they had no relevance to the research work undertaken. Most of the questions were designed to try and find out how close the implementation mirrored the required specification. Although it was felt that further more detailed questions about the reliability and maintainability of the software could have been asked, there was a limit on the time students could be expected to spend on this part of the exercise.

10.5 Experimental difficulties

This type of experiment can never fully replicate the conditions that exist when developing software in a commercial environment. Hence there are problems which will always affect the validity of the results.

Problems encountered included the following:

10.5.1 Developer teams not equally matched on ability

Although an attempt was made to create teams of equal ability, this will never be possible to achieve accurately in practice, hence the software produced by teams will always be influenced by ability as well as the actual development processes that the teams adopted. However from a practical perspective it would be almost impossible to set up this type of study with a large number of teams where it could be stated that each team of individuals was equally matched in terms of ability.

10.5.2 Personality and team co-operation

The way that a team operates in terms of its co-operation and structure will probably also affect the software produced and is very difficult to quantify and relate to the final product.

There is evidence that suggests that the communication processes (whether formal or informal) exert a very considerable influence on the performance of software developer teams. Evidence in the literature (Kraut and Streeter 1995, Curtis, Krasner and Iscoe 1988) has shown that research from organisational theory could well have more relevance to software development and software engineering than has previously been considered. Hence the inter-team dynamics will certainly have an affect on the

way that a team performs, although the mechanisms for measuring and monitoring this in any formal way are very difficult to set up.

10.5.3 Communication and prototyping

It was anticipated that most teams would use prototyping sessions as the best method to clarify queries about the requirement with the user. However unlike in a true commercial development most team members found it easier to communicate with the main user via electronic mail. This meant that the number of prototyping sessions that teams carried out was less than anticipated despite being encouraged to do so in classes.

As a result of this it was decided to group all communication about the specification irrespective of its source into one category, as well as looking at the results of the work solely concerned with prototyping. Hence in this study, communication via Email, prototyping sessions, phone calls or any other form are all considered in a similar fashion. The author acknowledges that in commerce the communication about the system is likely to take place in a more structured fashion and that perhaps electronic mail may not form such a major part of the communication process. Hence prototyping sessions and formal reviews would be more prevalent. However in this type of experiment it might be possible to do this, but unfortunately this would have to be built in at the start. Nonetheless the results from this experiment are unlikely to be significantly different because the main purpose of prototyping from a developers point of view is to communicate and learn more about the problem domain in order to build a system that reflects the users' true needs.

10.5.4 Use of undergraduate students in lieu of professional developers

It is recognised that the use of undergraduate computing students is likely to have an impact on the software development processes. For example, it is possible that the way that software is bought and sold might be influenced more by personality or friendship within the student cohort rather than the software itself. Hence the relationship between software quality and its popularity as a purchased product (i.e. sales) might be influenced by factors not as prevalent in an analogous commercial environment.

This cohort of students will also have substantially less experience of software development and hence a poorer knowledge of issues of quality, maintenance,

reliability and related factors. Hence the answers that they will have given to certain questions will be based on a more naïve understanding of software quality than a software professional of several years experience. This is also likely to have had an influence on the overall results. Unfortunately this type of problem is very difficult to make allowances for in this type of research project.

The two points above mean that the interpretation of the results presented here need to take account of these issues. Therefore the suggestion that similar or identical results are likely to be obtained from commercial programming teams with substantially greater experience and knowledge is not likely to be the case. It is important that these points are taken in to account when looking at the results presented.

10.6 Rating and analysis of software

Software produced by each group has been analysed and rated according to various criteria. These are as follows:

10.6.1 Dependent variables

This list comprises the dependent variables that have been analysed. I.e. those that have possibly been influenced by the groups behaviour, ability, or make up.

Sales made

This is a record of how many sales each group made at the software fair. It is therefore an fair indication of how the software was perceived by other people studying on the module. It is a valuable measure as it indicates the perception of peers as to the overall quality and likely maintainability of a piece of code. In effect it is a type of quality metric.

Conformance to specification

Each piece of software has been given a rating (out of seven) which considers how closely it conforms to the required specification. This is the *actual* specification required rather than the original high level written requirement. It has been calculated by assessing each system against a set of sample of questions and allocating a score for each correct implementation of the required specification.

Completeness

Each piece of software was assessed for completeness. This has been calculated in a similar fashion to the above variable. However in this case the completeness of the software has been measured by how far the group proceeded in the preparation of various reports that were required as the final part of the specification.

Reliability

This has been measured by assessing each system against a 3 point Likert scale. It has been collated from *at least* 3 different opinions from within the cohort of students studying on the module. It has been assessed by people using the system and recording its reliability when carrying out typical tasks.

Construction quality

This has been calculated in a similar fashion to the above, but investigates the general quality of the system in terms of the code construction.

Maintainability

Again as reliability, but considers the maintainability of the system.

Overall software quality

The 3 measures considered above can be combined to give an idea of overall system quality. This will yield a figure of between 0 (devoid of quality!) to 9 (apparently perfect).

10.6.2 Independent variables

The following section describes the independent variables that have been examined.

Prototyping sessions

If a team used prototyping as part of the exercise this was recorded

Number of queries about the specification (total communication)

The number of queries about the specification that each group made was recorded. Points about module administration and non-task specific enquiries were recorded but have been ignored in the analysis. Queries came either through Email, in meetings, or in prototyping sessions.

This information should give a good picture of how the groups went about clarifying the original specification.

10.7 Statistical analysis

The raw data for the analysis carried out is included in Appendix 2b.

The data is analysed in order to try and discover two sets of relationships.

a) Is there a significant difference in the performance of teams who used prototyping in developing their software compared to those that did not?

b) Is there a correlation between the volume of communication that took place between the development team and the main user?

At the outset of the study it was only intended to try and investigate point (a) above. However, as noted earlier it was felt that considerable communication that would

normally take place within prototyping sessions (as in a commercial environment) was conducted through other channels in this study. Hence it was decided to extend the analysis to discover whether level of communication also influenced the way that software was constructed.

A set of graphs are included, these show the relationship between various dependent and independent variables. It is important to note that the graphs show the raw data. (See Appendix 2b for details of data)

Hence the values of each of the following dependent variables have been examined with respect to the two questions under investigation.

- Sales made
- Conformance to specification
- Completeness
- Reliability
- Construction quality
- Maintainability
- Overall system quality

10.7.1 Analysis of performance of teams and the use of prototyping

It is important to choose a test that is suitable for the data being investigated.[3] With regard to this particular data set, the dependent variables are based on the aggregation of ranked data, hence the use of a parametric test would be inappropriate. As a result the data has been analysed using the Mann-Whitney *U* test.

[3] A comprehensive source of information is available in the Handbook of Parametric and Nonparametric statistical procedures by D.J. Sheskin (1997).

The hypotheses under review are based on the following statement:

Do the two samples represent populations with different rank-ordering of the scores for the dependent variables?

The calculations for the tests were carried out using SPSS for Windows, version 8.0.0.

It can be seen in table 10.1 that a total 22 teams were involved, 8 used prototyping, 14 did not.

Table 10.1 Mann-Whitney Test Results

	System prototyped	N	Mean Rank	Sum of Ranks
Sales made	**No**	14	10.86	152.00
	Yes	8	12.63	101.00
	Total	22		
Conformance to Specification	**No**	14	10.50	147.00
	Yes	8	13.25	106.00
	Total	22		
Reliability	**No**	14	10.39	145.50
	Yes	8	13.44	107.50
	Total	22		
Standard of construction	**No**	14	8.82	123.50
	Yes	8	16.19	129.50
	Total	22		
System completeness	**No**	14	9.93	139.00
	Yes	8	14.25	114.00
	Total	22		
Overall system quality	**No**	14	9.04	126.50
	Yes	8	15.81	126.50
	Total	22		
Maintainability	**No**	14	9.64	135.00
	Yes	8	14.75	118.00
	Total	22		

Table 10.2 Test Statistics for grouping variable - System Prototyped

	Sales made	Conformance to Spec	Reliability	Standard of construction	System completeness	Overall system quality	Maintainability
Mann-Whitney U	47.000	42.000	40.500	18.500	34.000	21.500	30.000
Exact Sig. [2*(1-tailed Sig.)]	.570(a)	.365(a)	.297(a)	.008(a)	.145(a)	.016(a)	.082(a)
(a) Not corrected for ties.							

From the table it can be seen that there are only 2 variables that have a significant level of difference between the two groups of teams. The standard of construction is significant at a 1% level. Overall system quality is the variable with the next largest difference in ranks between the two sets of groups, but it is significantly different at a 5% confidence level.

From the data supplied there is certainly no evidence to suggest that prototyping has had a major affect on the attributes of software development which are typically mentioned in the literature. In particular, conformance to specification seems to be very similar in both groups.

Given the above information it is very difficult to make any comprehensive statements about the effect of prototyping on the quality of software being produced, although there is some limited evidence (see above) that it does have some benefit.

As the outcome of this particular analysis was inconclusive and it had been noted that a great deal of communication had been logged outside of prototyping sessions, it was decided to look at the relationship between system development and the total volume of communication that had taken place both in prototyping sessions and outside of them.

10.7.2 Correlation between dependent variables and total group communication

The raw data for this section of the analysis is also shown in Appendix 2b in table A2. The figures shown in the table represent the total number of individual queries about the specification that were made by each team.

As in the previous section the type of data that had been collected dictated that a non parametric correlation test should be carried out. In this section the research is considering whether a correlation exists between the amount of communication that took place about the task and the value of the independent variable. Hence the most suitable tests to consider for this type of problem are Spearman's rank correlation coefficient (Spearman's rho) or Kendall's similar test (Kendall's tau,). Spearman's rank correlation coefficient is a more commonly used statistic[4] as it is considered to be more straightforward to calculate in most cases. However using contemporary statistical software this is not a major extra problem, hence in this work both statistics have been calculated. It is worth noting that because of the logic involved that the figure for Kendall's tau will normally be lower than for the equivalent Spearman value, nonetheless both are equally likely to detect a significant effect in the population.

It should also be noted that the group with the largest figure for total communication (group M12) had a figure that was well over double that of the next group. Hence this particular figure could well be classed as an outlier and could be rejected from the analysis. However as the test used was non parametric, its inclusion is not likely to create a major distortion of the results, hence initially it has been left in the analysis presented here.

The set of graphs presented here show the relationship between the dependent variables and the independent variable "total communication". They are based on the ***raw data*** held in table A2 and as such do not show the ranked data. However the analysis and calculations have been carried out on the ***ranked data*** rather than these figures. The graphs are included to show the pattern of the raw data.

[4] An explanation of this is provided in Sheskin (op.cit)

Graph 10.1

This graph plots sales made (y-axis) and total communication (x-axis).

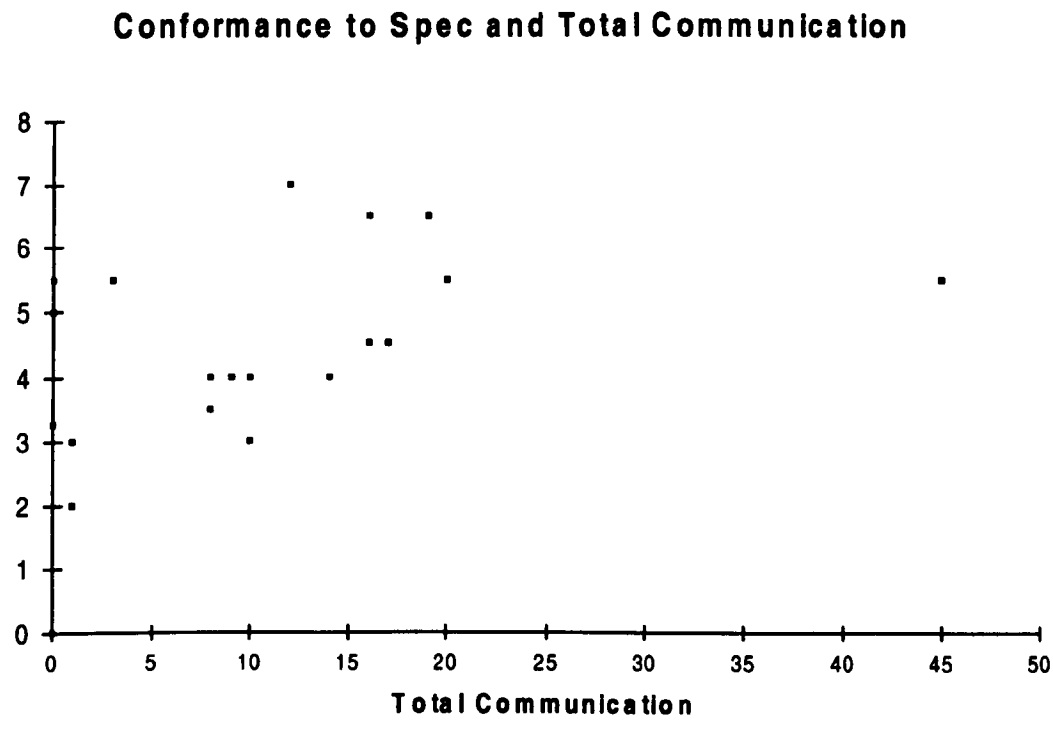
Sales made (ranked)

Kendall's tau Total Comm (ranked)	Correlation Coefficient	.427(**)
	Sig. (1-tailed)	.006
	N	22
Spearman's rho Total Comm (ranked)	Correlation Coefficient	.542(**)
	Sig. (1-tailed)	.005
	N	22

** Correlation is significant at the .01 level (1-tailed).

There is a moderately good correlation between total communication and sales made. This is significant at a 1% level of confidence for a one tailed distribution. Hence it is reasonable to conclude that the number of sales made of the software product has some relationship to the amount of communication that took place between development teams and the main user.

Graph 10.2



This graph plots sales conformance to specification (y-axis) and total communication (x-axis).

Conformance to Spec(ranked)

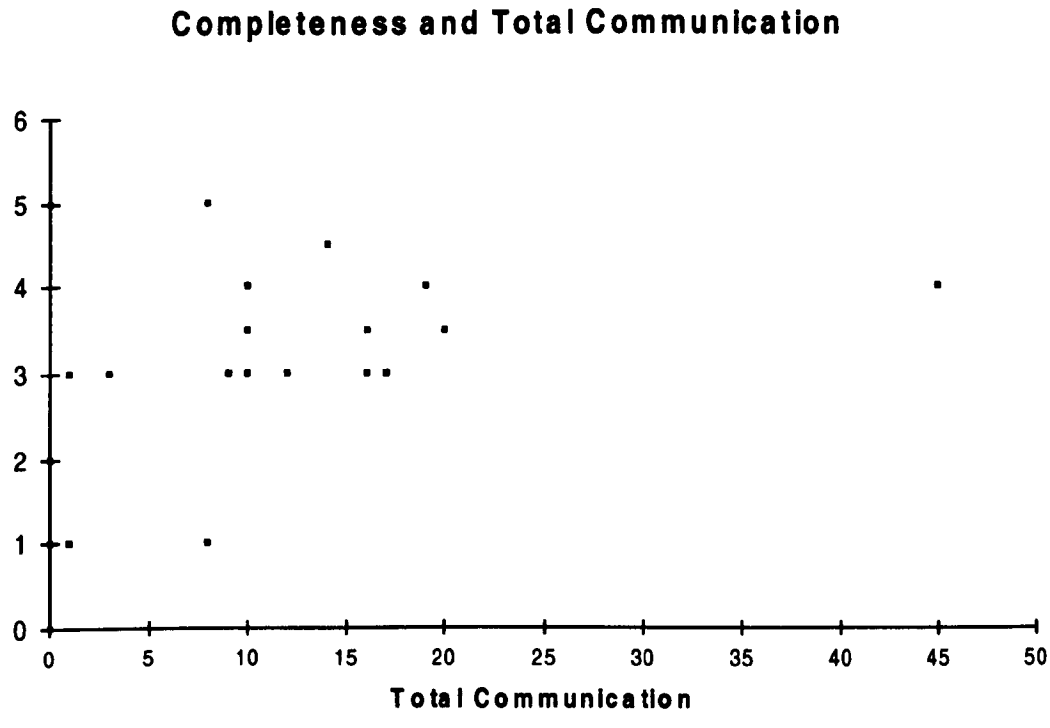
Kendall's tau Total Comm (ranked)	Correlation Coefficient	.435(**)
	Sig. (2-tailed)	.008
	N	22
Spearman's rho Total Comm (ranked)	Correlation Coefficient	.550(**)
	Sig. (2-tailed)	.008
	N	22

** Correlation is significant at the .01 level (2-tailed).

There is a fairly strong correlation between the level of communication and the conformance of the software product to the required specification. It can be seen that there are two groups who score zero (M19 and T01). These represent no working implementation. The time scales imposed were tight, hence failure to deliver a working piece of software suggests pressures akin to those that occur in commercial development. Beyond that point the groups are arranged on a scale running from approximately two up to seven. It is interesting to note that a total of fourteen groups

score under five. This suggests that most did not produce a piece of software that was particularly close to the actual specification required. Nonetheless there is some evidence that the groups who communicated more about the specification produced better "fitting" software. However there are also groups who did not communicate at all (M3 and M4) but still produced software that scored above average scores. The possible reasons for this are discussed in the summary at the end of this section.

Graph 10.3



This graph plots completeness of the implemented system (y-axis) and total communication (x-axis).

System completeness

Kendall's tau Total Comm (ranked)	Correlation Coefficient	.425(*)
	Sig. (2-tailed)	.010
	N	22
Spearman's rho Total Comm (ranked)	Correlation Coefficient	.516(*)
	Sig. (2-tailed)	.014
	N	22

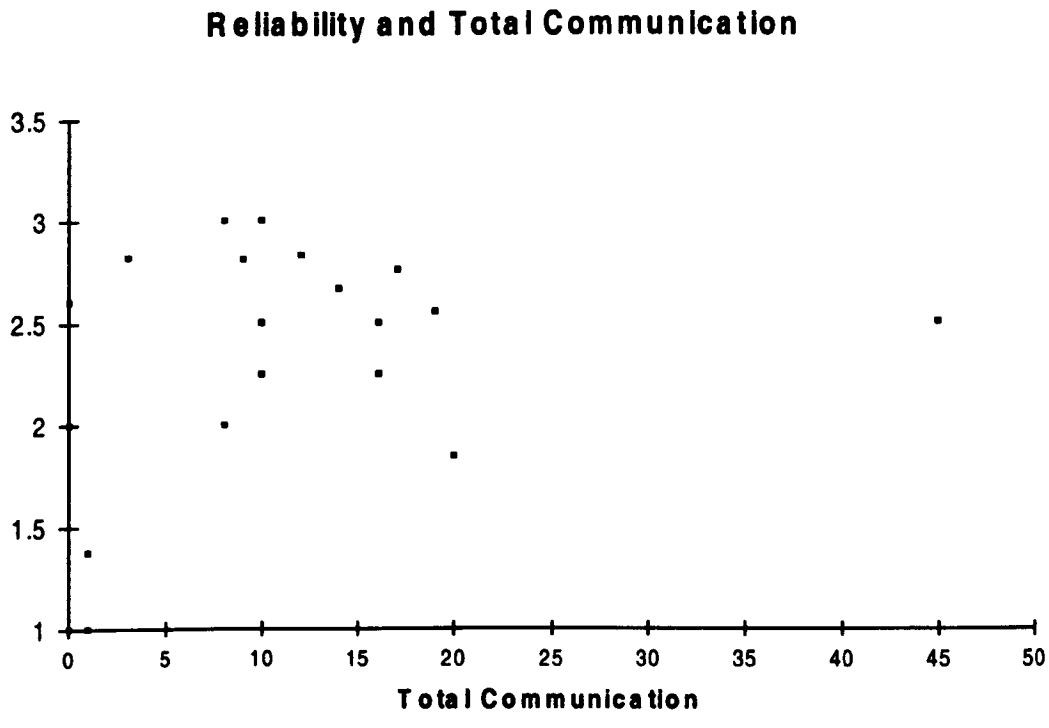
* Correlation is significant at the .05 level (2-tailed).

There is a moderately strong correlation between the level of communication and the conformance of the software product to the required specification. However although it is statistically reasonable to accept that there is a fair degree of confidence in the correlation, for the reasons given below it is suggested that this analysis may be flawed.

The measure of completeness of implemented specification uses the number or reports produced by every team as a measure. Unfortunately, analysis of the results showed that several teams produced more reports than were specified! The maximum required was three, several teams produced four or even five! As a result the statistical analysis of these results will be affected. However it is certainly interesting to note that all of the teams (six in total) that produced fewer than the required number of reports fell into the lower half of the sample with regard to total communication.

The fact that a significant minority of the teams produced greater than the maximum required number of reports could suggest that the original specification was not particularly clear. Hence the analysis is based on the actual number of reports produced, rather than an adjusted figure, even though an adjusted figure might offer "better" statistical results.

Graph 10.4



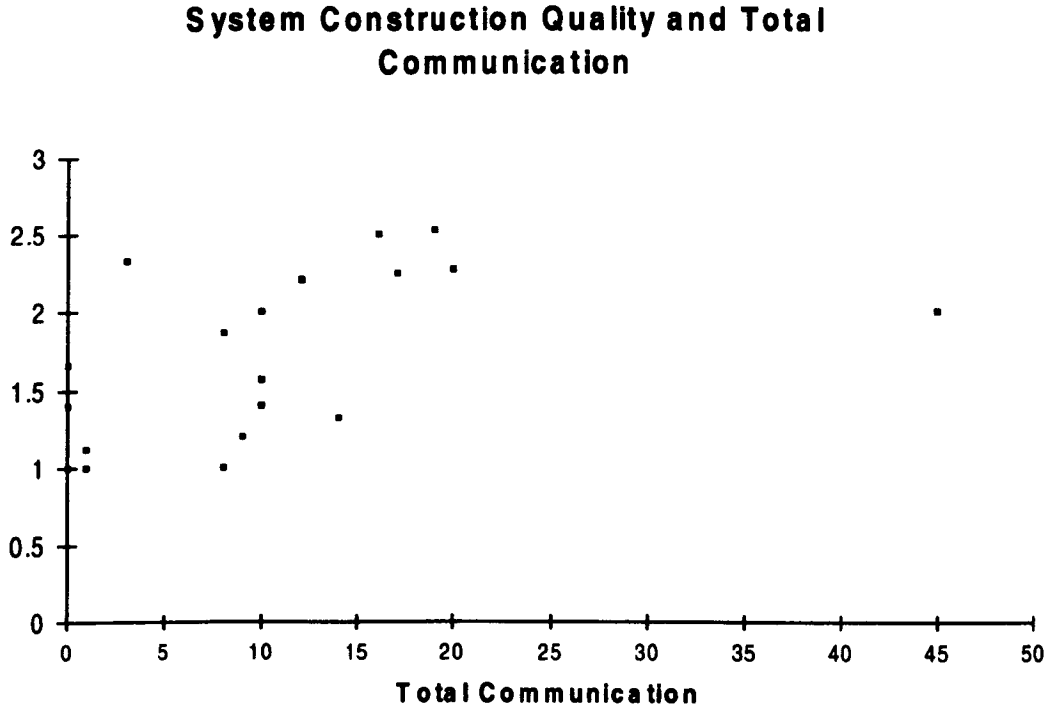
This graph plots system reliability (y-axis) and total communication (x-axis).

Reliability (ranked)

Kendall's tau Total Comm (ranked)	Correlation Coefficient	.230
	Sig. (2-tailed)	.153
	N	22
Spearman's rho Total Comm (ranked)	Correlation Coefficient	.357
	Sig. (2-tailed)	.103
	N	22

There is no statistically significant correlation between the reliability of the systems produced and the amount of communication that took place between developers and the main user.

Graph 10.5



This graph plots overall construction quality (y-axis) and total communication (x-axis).

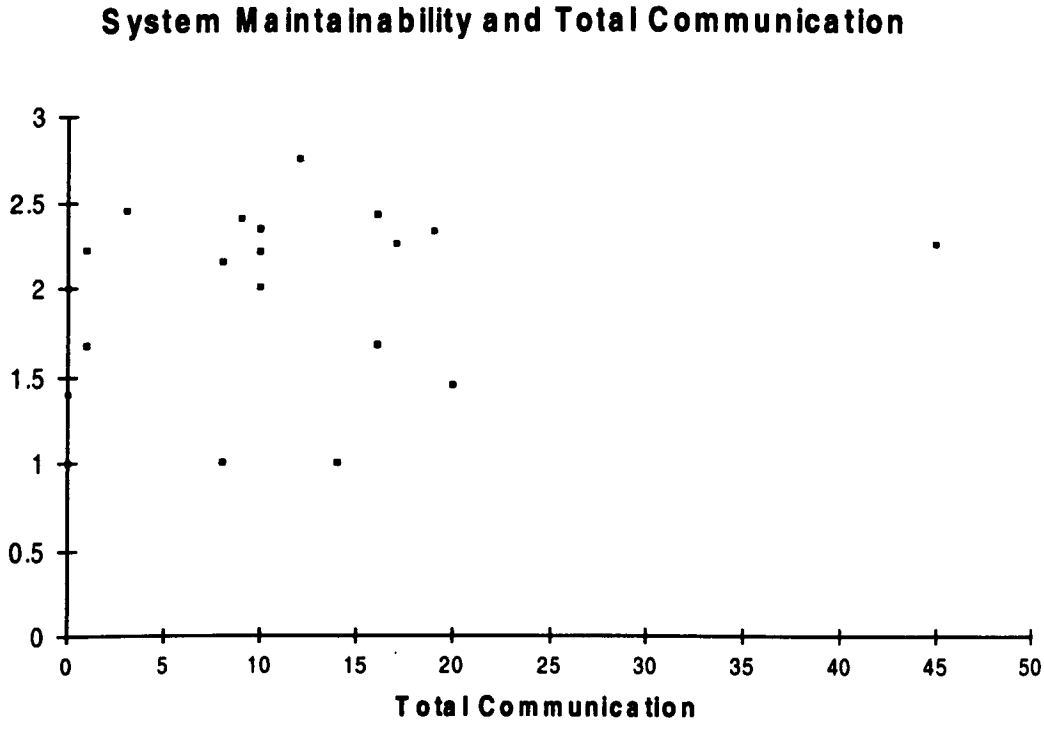
Standard of construction

Kendall's tau Total Comm (ranked)	Correlation Coefficient	.554(**)
	Sig. (2-tailed)	.001
	N	22
Spearman's rho Total Comm (ranked)	Correlation Coefficient	.712(**)
	Sig. (2-tailed)	.000
	N	22

** Correlation is significant at the .01 level (2-tailed).

There is a strong correlation between the level of communication and the standard of construction of the system. This is a slightly puzzling relationship, as there is no obvious reason why better communication should lead to better code construction. It is possible that this result is spurious in that is likely that better teams tend to communicate more and also tend to build better systems. Hence the explanation of this apparent relationship is that it is a measure of correlation between system quality and group ability rather than simply communication and system quality. However this is only speculation.

Graph 10.6



This graph plots maintenance (y-axis) and total communication (x-axis).

Maintainability (ranked)

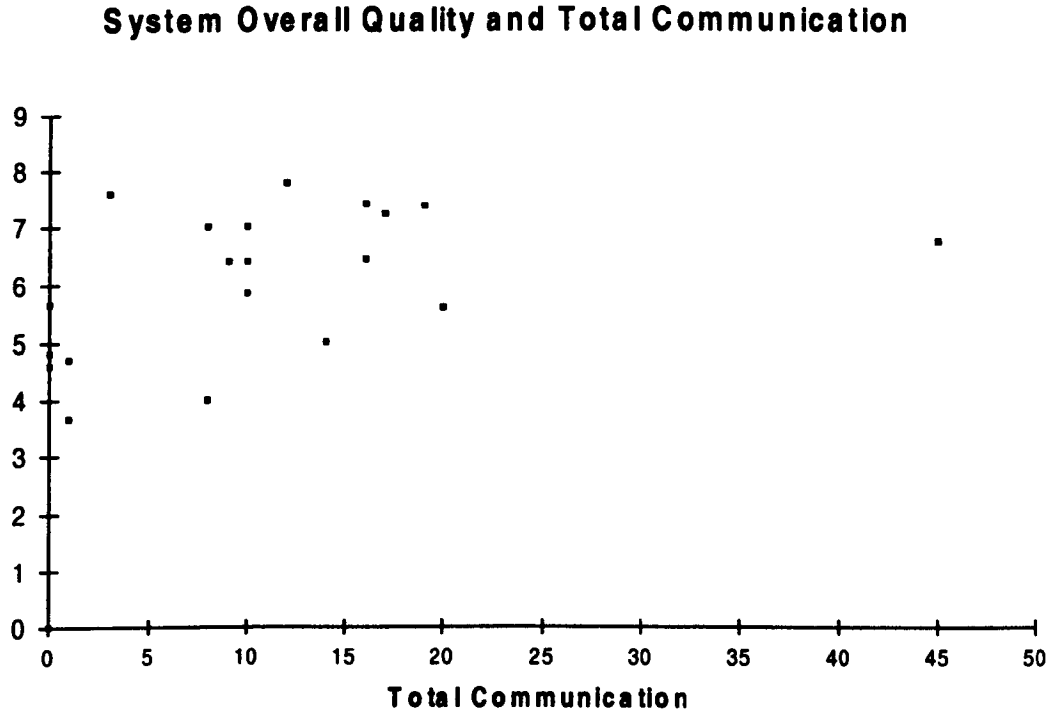
Kendall's tau Total Comm (ranked)	Correlation Coefficient	.300
	Sig. (2-tailed)	.062
	N	22
Spearman's rho Total Comm (ranked)	Correlation Coefficient	.426(*)
	Sig. (2-tailed)	.048
	N	22

* Correlation is significant at the .05 level (2-tailed).

There is evidence of a limited relationship between the level of communication and the perceived maintainability of the software product. However the two tests fall either side of the 5% confidence limit which suggests that the confidence in this case is marginal. This is interesting in that the correlation is much lower than in the previous case.

Hence a positive perception about the quality of construction of a system is not immediately transferable to maintenance. The concepts of "good quality" and "maintainable" do not seem to be entirely interchangeable.

Graph 10.7



This graph plots overall quality (y-axis) and total communication (x-axis).

Overall system quality (ranked)

Kendall's tau Total Comm (ranked)	Correlation Coefficient	.442(**)
	Sig. (2-tailed)	.005
	N	22
Spearman's rho Total Comm (ranked)	Correlation Coefficient	.614(**)
	Sig. (2-tailed)	.002
	N	22

** Correlation is significant at the .01 level (2-tailed).

There is a strong correlation between the level of communication and the overall quality of the software product. As with the standard of construction it is not immediately obvious why the level of communication should be related to the final quality of the product. It may well be that better teams communicate more, hence again this is more a measure of the competence of the team, rather than the level of communication.

10.7.2 Summary of specification query results

From the analysis carried out, it can be seen that the dependent variables have varying degrees of correlation with the number of communication queries raised.

The most strongly correlated is the standard of construction of the system, followed by overall quality, then conformance to specification, sales made, completeness; and finally maintainability, which appears to have a limited relationship only. There was no significant correlation between level of communication and system reliability.

In the view of the author it is difficult to understand why the number of queries should be directly related to the construction quality of the system. It is possible that *better* systems have been constructed by people who also took the time and effort to query the original requirements as the system was under construction. Hence the correlation is not due to the actual communication, but due to better software developers writing better systems and actually finding out more about the specification as well.

The second highest strongest correlation (overall system quality and communication) is an aggregated score which includes the standard of system construction, hence it is not that surprising that it should have a fairly high correlation.

In the opinion of the author it is also slightly surprising that the correlation between conformance to specification and communication is not higher. It seems strange that some groups managed to build software that conformed reasonably well to the specification without much communication occurring. This suggests that either:

- Groups found out about the specification via other means (i.e. from other people)

- There is a maximum sensible level of communication, above which no real extra value about the specification is learnt.

Finally the lack of strong correlation between maintainability and communication and the lack of correlation between reliability and communication seems to contradict some of the findings made by Boehm in 1984. However it does not seem particularly surprising to the author that systems that are reliable and maintainable should by necessity have had a lot of communication take place about their specification between developers and clients. A system may be both reliable and maintainable, but may still not do what is required of it by a client!

10.8 General summary of experimental results with conclusions

The analysis of the of data yields some interesting results, but some caution is required when interpreting it.

The direct impact of prototyping on the systems built does not appear to be particularly significant. There are various reasons why this could be the case:

- There is actually no direct relationship between prototyping and the quality of systems built

- There is the possibility that the added complication of additional changes required because of the use of prototyping could have put off the relatively inexperienced development teams

- The experiment as set up and administered had errors which led to an inaccurate conclusion being reached.

In the opinion of the author, there was an unexpectedly high level of communication that took place between development teams and the main user outside of the proposed main channel of communication i.e. the prototyping sessions. However the use of electronic mail could well have had an effect on the use of prototyping and hence lessened the direct impact of prototyping on the systems developed. As a result the author feels that the explanation for the results obtained are more likely to be as a consequence of deficiencies in the way that the original study was set up.

The second part of the analysis using total communication as an independent variable provides some evidence that the statement above could be a reasonable explanation for the initial results obtained.

On initial examination there is a good correlation between levels of communication and various aspects of systems development. This suggests that higher levels of communication do lead to better systems being developed. Despite this evidence it is the opinion of the author that a range of other factors could well be influencing the results obtained and therefore it would certainly be risky to draw the simple conclusion that increased communication between developers and users leads to better systems. It is likely that other aspects of group dynamics, plus group ability, the

experience of development teams, as well as factors such as system design methods and tool usage could also have a significant effect on systems being built.

It is the opinion of the author that good groups by their very nature do build better software and part of the reason for that is that they do communicate effectively with their users and clients. However in addition it should be said that the process of software development is complex and that in order to isolate out the most significant factors further studies would require to be undertaken. As stated at the outset of this chapter the sample of students used was divided up in order to try and create groups equally weighted on ability. Unfortunately it is very difficult to accurately ascertain whether this had the desired effect on the cohort.

It is the opinion of the author that communication is a very important part of systems development, but that it probably is more effectively carried out by more able or more highly motivated individuals. Hence this seems to echo the evidence of DSDM consortium members who focus on the need for highly skilled technical development teams who are also capable of effective communication with clients and user groups.

In contrast to the above there is very little evidence to back up the claims made by other published work which suggests that software built using high levels of prototyping and communication is more reliable and maintainable. The experiment carried out was set up to encourage teams to build software that was likely to be perceived as maintainable and reliable by their peers. However analysis of the results indicates that these attributes are not correlated to communication or group ability to any significant degree. It is likely that teams felt that achieving success at the software fair (i.e. selling several copies of their software) was not likely to be compromised by software that had faults in it which were likely to lead to reliability or maintainability problems in the future. Hence they put more effort into producing structured code which appeared to be maintainable and in addition tried to conform to the real requirements of the user.

Chapter 11. Conclusions

This research project has investigated both the positive and negative aspects of prototyping within the development of systems using iterative and evolutionary methods. During the time that the work has been carried out it appears that RAD with a heavy emphasis on prototyping has started to become a respectable form of development (E.g. Martin 1991, DSDM 1995a and DSDM 1995b) and that since the conception of the DSDM consortium more commercial developers are using RAD style techniques.

Nonetheless it is apparent from this research that merely dressing a project up in the clothes of a RAD style development method, or the use of some form of prototyping in an ad hoc fashion within a project will not guarantee success. Commercial software developers must be careful not to fall in to this trap.

11.1 Summary of main findings

The original research aims and objectives are reiterated below (in italics) and the findings of the project are compared against them. Comments on the effectiveness of the research methods used are also listed.

- *As part of the literature review one of the initial aims of the research was evaluate the similarities and differences between the various surveys carried out in the subject area and to discover whether any particular recurring themes appeared in the results reported by them. In order to carry this out it was decided to examine the published survey evidence that had been undertaken in both in the United Kingdom and North America.*

A range of survey evidence was examined as part of the project (see Chapter 4). The analysis of the surveys showed that prototyping was extremely important as a vehicle for communication and learning. The area where prototyping seemed to have problems was its high use of users (and developers) time. From this it is suggested that developers and users using prototyping ensure that mechanisms are in place that allow for the best utilisation of the time allowed to take place.

Comment on using survey material

The use of survey evidence can lead to misleading information being obtained because of problems associated with the technique (see Chapter 1). In particular there are particular issues to do with self selection of the sample, and the difficulty in finding out the reasons behind the answers given. However the analysis of the survey data carried out as part of this project does seem to suggest that consistent recurring themes occur in most of the survey work examined. Hence in the opinion of the author this particular technique has yielded useful information which software development practitioners could learn from.

- *An investigation in to the effectiveness of prototyping techniques and an examination of the usefulness of control mechanisms that could be applied was undertaken. Hence an exercise in action research where several pieces of software were written using a variety of prototyping techniques in a commercial development environment was carried out.*

The case study evidence carried out as part of this research suggest that a high degree of user participation will ensure that the system requirements of a client organisation will be more closely met if the communication mechanisms are effective. This is undoubtedly an extremely important positive aspect of prototyping, and has been apparent in most aspects of the work carried out in the study.

The experiences of the author during the work undertaken for the action research suggest that using prototyping does have a major impact on the acceptability of the systems produced. The case study evidence also suggests that highly user centred systems development is not necessarily difficult to manage. This contradicts some of the (mainly anecdotal) evidence suggested in the literature. However, there is some evidence that effective control mechanisms must be agreed and in be place prior to the project starting if failure is to be avoided. The failure of the largest development at Frost Electroplating illustrates that the adoption of prototyping alone does not guarantee success. Hence development frameworks such as the DSDM initiative, and the TPM model proposed by the author should allow project managers to be able to set up and control projects in an effective fashion. The models can be customised to the needs of the organisation.

- *Research in to the applicability of function point analysis to discover how useful this particular technique could be in helping to control projects involving a high*

degree of prototyping was undertaken. The results of a rapid software development project where function points had been used as part of the project management process were analysed in order to investigate this technique.

From the work carried out it appears that the use of function points for estimation purposes has questionable benefit when combined with a highly iterative and changing project environment. It is acknowledged that this did not form a major part of the work, and that further research is required in this particular area, nonetheless the enthusiastic adoption of this method by RAD and prototyping practitioners could be misguided.

Comment on using case study and action research work

The opinions of the author on using case study and action research are very much in line with the advantages and disadvantages by Galliers (1994). There is no question that the results from the work carried out both in the building of systems and the use of function point analysis are only applicable to the conditions that existed at the time of the work being carried out and within the organisations concerned. Hence it is not valid to try and draw more general inferences from those results. However detailed information about the way that prototyping, incremental development and issues in project management could work in practice can be examined in fine detail, which can be useful to other researchers in similar though not identical circumstances.

The problems of having no influence over certain environmental factors and no control study to compare results against, together with the inevitable subjective interpretation of the results also have an impact on the usefulness of findings from such studies.

Overall it seems reasonable to conclude that the work can be of some value to other researchers as long as these drawbacks are taken in to account. They cannot be regarded as "typical" in any way as they are isolated studies and, as stated above, general inferences should not be draw.

- *The research aimed to discover how a rapid software development project would affect the production of software in terms of its overall functionality and quality and how the use of prototyping in particular affected the development process. Hence an experimental study where software development teams produced*

working implementations of a system specification was set up. The teams could use a model of development that is proposed in this project.

The experimental evidence produced from the final part of the investigation did not produce the results that were anticipated. It had been expected that the use of prototyping would have a major impact on various aspects of the systems that were developed. In fact this was not the case. Hence from the work carried out here the most sensible conclusion that can be drawn is that the use of prototyping does not have a particularly major impact on systems developed. However this contradicts the findings of the author in other parts of this project and also the findings of most other researchers in the field. In the view of the author it is likely that the results from this particular part of the work should be treated with extreme caution.

The second part of the experimental analysis examined the wider field of total communication (including the use of prototyping). This provided contrasting results to the other experimental work.

The results from this analysis suggest that the level of communication that takes place between users and developers can have a varying effect on several aspects of systems development. Hence it is the view of the author that high levels of communication should be encouraged in systems development in order to help build quality systems that conform to the users' real needs.

The experimental work undertaken suggests that the structure of a system can be influenced by good communication, but that this *could* be a by-product of the overall ability of a particular systems development team. Nonetheless project managers should instil the necessity of communicating properly with users into project teams.

In order for commercial developers to gain an advantage from using a development approach including prototyping it is most important that they also have other quality control mechanisms built in to the project. Quality will not automatically come about from using prototyping.

Comment on using experimental evidence.

The results and subsequent analysis of the experiment indicate that there were certain unforeseen circumstances that had not been predicted at the outset. In particular it had been anticipated that the bulk of the communication about the problem specification

would take place during the prototyping sessions. In the experiment this did not prove to be the case. A very high level of communication took place outside the prototyping sessions, with the use of electronic mail proving to be very popular with the teams involved.

In addition to this it is very difficult to speculate on how the developments would have progressed if:

- Prototyping was made mandatory for all teams
- Prototyping was made illegal for all teams

Although it could be possible to replicate the actual system specification, it is not possible to use the same development teams again, so the effect of using different personnel would need to be taken in to account if a repeat of the exercise was carried out. This is a problem cited in Chapter 1 as a general issue in this type of research and there is no doubt that the problem existed in this project.

Another drawback of this form of research is the problem of how realistic the small-scale experiment is compared to projects in commercial environments. Although some of the environmental circumstances can be replicated to a certain extent, it is not possible to confidently state that the results from this work can really be applicable to large commercial projects.

A further problem that may well have influenced the outcome of the research is that the teams could not be selected with identical skill and motivation levels at the start of the experiment. Although an exercise was carried out to try and ensure that every team had a similar profile, there is no doubt that some teams may well have been technically more competent than others. Although this does not invalidate the findings entirely it should be noted as a problem.

However it should also be acknowledged that despite these drawbacks, this form of research can yield useful information if the experiments are set up and run carefully and the analysis takes in account the drawbacks mentioned above.

Objectives

The overall objectives of the research project are to assess the value of rapid software development and prototyping.

The four aims listed on the previous page were formulated in order to try and find out whether: recurring themes about the use of prototyping occurred in the survey literature; how useful prototyping can be in conjunction with various control mechanisms; if function point analysis and prototyping can be used in conjunction to provide useful project management information and finally to discover what influence rapid software development and prototyping have on the development of software.

The project has attempted to evaluate the value of software prototyping using the research methods outlined above. The survey evidence (reported in Chapter 4) did suggest some recurring themes commented on by different researchers. The results of the practical work undertaken suggest that it can have a positive impact on systems development. The (enforced) use of TPM methods in the experimental work showed that most teams could produce software within the required time frame. However the fact that some elected to use prototyping whilst others did not also suggests that in reality issues in project management are similar in most systems being developed. If prototyping is used, there is no apparent reason why that project is more difficult to control. The work at Frost Electroplating and the final experimental part of the research suggest that such projects can be controlled, but as illustrated by the project failure at Frost Electroplating appropriate control mechanisms must be in place to ensure success. The TPM model outlined is similar to the type of method suggested by the DSDM and in the view of the author, high level customisable methods are preferably to very prescriptive mechanisms that may not be appropriate for some projects or organisations. The use of function point analysis was investigated (albeit to a limited extent). From the work carried out it seems that using prototyping and function point analysis together does not seem to enhance the project management process greatly, however it is acknowledged that further work is required in this area. Overall from the work carried out in this project there is some evidence that prototyping and rapid software development can have an influence on software developed, but that some claims made for it are exaggerated.

11.2 Contribution of this work to the body of knowledge in the area

- The investigation suggests that prototyping is not by itself a panacea to the problems of delivering quality software on time and on budget. The work carried out for the metal finishing company (outlined in Chapters six and seven) although using prototyping to a high degree was not always a success and although some parts were delivered within the anticipated time frame, this was not always the case.

- Some claimed advantages of prototyping do not appear to be supported by the work carried out in this particular study. Previous work carried out in the field (e.g. Boehm 1984) suggests that using prototyping has a significant effect on several aspects of software quality. The experimental work carried out within this research project does not entirely back this up, suggesting that some of the claims made require further work in order to justify them.

- Prototyping managed within an appropriate framework can help deliver quality software on time and within budget. In contrast to the first point in this list there was evidence in the work carried out for the metal finishing company that using incremental development and delivery could lead to systems being delivered in a satisfactory fashion for clients. However the work carried out as part of the experimental work in Chapter 10 did not allow this to be investigated in depth. Hence although the TPM approach was followed to a certain extent by some teams, and the DSDM model appears to becoming more widely adopted. It is not possible from the work carried out here to conclude that these mechanisms will always help in the delivery of systems on time and within budget.

- The use of estimating methods for iterative projects where change is anticipated and encouraged is not entirely beneficial and requires further investigation (Chapter 8). The analysis of the function point data suggests that its use in conjunction with prototyping has only limited usefulness. At a high level it might be useful as an estimation aid to a multi-increment project, however as a technique for monitoring and estimation at a lower level the evidence suggests that it is of little value. However this is an area which could certainly benefit from further more extensive research work being carried out.

11.3 Further areas of possible research

The work carried out does suggest areas where further work would be useful.

11.3.1 Estimation and Planning

In 1987 Mayhew identified that planning was an area which required further investigation. Despite the wealth of published material since then, this is an aspect of RAD style development that still has considerable uncertainty associated with it. In the view of the author the adoption of a function point or similar method, by the DSDM consortium will not prove to be a particularly beneficial step. The evidence presented in this work suggests that although it might have some value in assessing productivity of projects that are already underway, the use of such a method at the inception of a RAD project is limited. In addition the progress of this type of development can not be adequately controlled at a micro level using function points. This certainly requires further work. As RAD projects encourage change and iteration, the value added as parts of a system are changed or deleted will not be adequately represented using function point analysis.

11.3.2 Documentation

Adequate documentation of systems developed using evolutionary and incremental methods is an area that is not covered well in the literature. There is a conflict between updating and creating new documentation to support changing systems, and carrying on with further system development to meet proposed deadlines. The TPM model proposed earlier (see Chapter 6) emphasises the importance of the documentation process, but the author feels that further work is required to give commercial developers more insight in to the most effective way of documenting such systems. It is revealing that the DSDM manual does not have a section devoted to documentation listed at all.

11.3.3 Contractual Issues

The failure of systems developers to deliver quality products on time has become a matter of even greater significance of late with the judgement of the High Court in the case of St Albans Council Vs ICL as reported in the computing press (E.g. Collins 1996, Boarman 1996). For developers engaged in RAD style development the

problems associated with incremental and evolutionary development could complicate matters further. There are a variety of particular problems that need to be addressed:

- User responsibility: If the users are handed a more central role in both the development of the software, and also the decision making processes, then they should be expected to shoulder more responsibility for the success or failure of a project. It is likely that highly motivated users with a real desire to see systems built that will help them, will take on this role in a sensible fashion. However there is also the possibility that under certain circumstances (e.g. pressure from other areas of the workplace, conflict with developers, users not available) the system will fail to be delivered because of inadequate or inappropriate input from the user/client group. This could well have repercussions from a contractual point of view.

- Incremental and evolutionary delivery: The very nature of "partial" delivery of systems is a complex area. It is well documented that large monolithic developments are very vulnerable to delivery problems because of their size and complexity. RAD projects aim to get around this by delivering solutions within a short time scale. This means that the scope of a project must be very carefully mapped out, and that it is likely that increments could be small compared to traditional software systems. Kerr and Hunter (1994) illustrate this as follows:

  > "Be ruthless when it comes down to paring down the scope of a project..."

However it is very important that the customer is in agreement with the idea that the system that they want will be delivered in a series of stages, or evolves over a period of time. Failure by the developers to ensure that customers understand this will lead to problems. It is important that when the contractual and planning stages of a project are being considered, these issues are taken in to account.

11.3.4 Quality of systems delivered

The experimental work carried out in the research did not back up the earlier findings of Boehm (1984) with regard to the quality of delivered software. There are various reasons why this might have been the case, but irrespective of these results, system developers must be careful not to compromise quality when developing RAD solutions. The whole area of quality and its measurement in system development is

complex and it could be argued that if time to delivery is the most important metric for a particular development, then RAD will almost by definition deliver higher quality software. This is a rather superficial view and as the experimental work suggests, software quality will not automatically come about as a by-product of the RAD development process, but needs to be controlled or engineered in to software by using appropriate quality processes. The identification of the most effective mechanisms for doing this in a RAD project is an area that is worthy of further investigation.

Bibliography

Alavi, M. (1984) An assessment of the prototyping approach to information systems development. *Communications of the ACM.* Vol 27, No 6.

Albrecht, A. J. Gaffney, J. E. (1983) Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation. *IEEE Transactions on Software Engineering.* Vol 9, No 2.

Arthur, L. J. (1992) *Rapid Evolutionary Development: Requirements Prototyping & Software Creation.* John Wiley.

Balzer, R. (1988) Draft report on requirements for a common prototyping system. *SIGPLAN Notices.* Vol 24, No 3.

Bansler, J. P. Bodker, K. (1993) A Reappraisal of Structured Analysis: Design in an Organizational Context. *ACM Transactions on Information Systems.* Vol 11, No 2.

Basili, V. (1985) Quantitative evaluation of software engineering methodology. *Proceedings 1st Pan Pacific Computer Conference* Melbourne, Australia, September 1985.

Bates, P. E. (1989) Rapid Software Prototyping: An introduction and motivation. *2nd BCS Seminar series on new directions in software technology* Wolverhampton Polytechnic, Wolverhampton, Spring 1989.

Baum, D. (1992) Go totally RAD and build Apps faster. *Datamation* Vol 38, No 19.

Baümer, D. Bischofberger, W. R. Lichter, H. Züllighoven, H. (1996) User Interface Prototyping - Concepts, Tools and Experience. *Proceedings of the 18th International Conference on Software Engineering.* Berlin. Germany. March 1996.

Bean, J. E. (1988) *Finishing Handbook and Directory.* Issue 38. Sawell Publications Limited.

Beckett, H. (1994) Life in the Fast Lane. *Computing.* September 22nd 1994.

Bicknell, D. (1994) Users deliver standard for rapid development. *Computer Weekly.* October 27th 1994.

Boar, B. H. (1984) *Application Prototyping a requirements definition Strategy for the 80's,* John Wiley, Chichester, UK.

Boarman, C. (1996) A cautionary tale. *Computer Weekly* September 26th 1996.

Boehm, B. W. (1981) *Software Engineering Economics.* Prentice-Hall.

Boehm, B. W. (1986) A spiral model of software development and enhancement. *ACM SIGSOFT Software Engineering Notes.* Vol 11, No 4.

Boehm, B. W, Gray, T. E. and Seewaldt, T. (1984) Prototyping versus Specifying: A multiproject experiment. *IEEE Transactions on Software Engineering,* Vol 10, No 3.

Boehm, B. W. Papaccio, P. N. (1988) Understanding and Controlling Software Costs. *IEEE Transactions on Software Engineering.* Volume 14, No 10.

Booch, G. (1994) *Object oriented analysis and design with applications.* 2nd Edition. Benjamin/Cummings. Redwood City, California, USA.

Braunstein, D. N. Lauer, T. W. Doane, D. P. (1991) Information requirements prototyping for decision support systems. *Journal of Information Technology.* Vol 6, No 1.

Bray, P. (1993) Killer Applications. *Which Computer.* January 1993.

Brimson, J. A. (1988) Bringing cost management up to date. *Manufacturing Engineer.* Vol 100, No 6.

Brittan, J. N. G. (1980) Design for a changing environment. *The Computer Journal.* Vol 23, No 1.

Brooks Jnr, F. P. (1987) No Silver Bullet. Essence and Accidents of Software Engineering. *Computer.* Volume 20, No 4.

Brown, D. W. Carson, C. D. Montgomery, W. A. Zislis, P. M. (1988) Software Specification And Prototyping Technologies. *AT and T Technology Journal.* Vol 67, No 4.

Budde, R. Zullighoven, H. (1990) Prototyping Revisited. *Compeuro '90 Proceedings of the IEEE 1990 conference on Computer Systems and Software Engineering.* Tel Aviv Israel.

Buckley, M. Candy, L. Edmonds, E. A. (1988) Determining Requirements and Prototyping the User Interface Module. Methodology for Expert System specification. *Human and Organisational Issues of Expert systems.* Stratford. UK. May 1988.

Bull, R. J. (1990) *Accounting in Business.* 6th Edition. Butterworths. London, UK.

Butler, A. and Chamberlin, G. (1987) The Aries Club-Experience of Expert Systems in Insurance and Investment. *Proceedings of the 7th Technical Conference of the BCS Specialist Group on Expert Systems.* Edited by M. Merry. Cambridge University Press.

Carey, J. M. (1990) Prototyping: alternative systems development methodology. *Information and Software Technology.* Vol 32, No 2.

Checkland, P. Scholes, J. (1990) *Soft systems: methodology in action.* John Wiley and Sons.

Cohen, B. Harwood, W. T. Jackson, M. (1986) *The Specification of Complex Systems.* Addison-Wesley.

Collins, T. (1996) Gamble of the century. *Computer Weekly.* August 1st 1996.

Connell, J. and Shafer, L. B. (1989) *Structured rapid prototyping.* Yourdon Press Computing Series.

Cook, S. Daniels, J. (1994) *Designing object systems: object-oriented modelling with Syntropy.* Prentice Hall, New York, USA.

Cornes, R. (1990) Method with just a touch of Madness. *The Guardian.* January 25th 1990.

Cotterman, W. Kumar, K. (1989) User Cube: A Taxonomy of End users. *Communications of the ACM,* Vol 32, No 11.

Cowderoy, A. J. C. Jenkins, J.O. (1991) Estimation using analogy. *4th BCS Seminar series on new directions in software technology* Wolverhampton Polytechnic, Wolverhampton, Spring 1991.

Crinnion, J. (1992a) The evolutionary development of business systems. *Proceedings of the IEE Colloquium on Software Prototyping and Evolutionary Development* London. November 11th 1992.

Crinnion, J. (1992b) Exploitation of the 4GL. *Software Devlopment '92-Management Track* Blenheim Online, London.

Cross, S. Estrada, R. (1994) DART: An Example of Accelerated Evolutionary Development. *Proceedings of the 5th IEEE International Workshop on Rapid System prototyping.* Grenoble. France. June 20th-23rd 1994.

Davis, A. M. (1990) System testing: implications of requirements specifications. *Information and Software Technology.* Vol 32, No 6.

Davies, A. M. (1992) Operational prototyping: A new development approach. *IEEE Software* Vol 9, No 5.

Davies, P. (1993) *An Investigation of Rapid Software Prototyping in a Commercial Environment and the Management Issues Raised.* BSc Project Report. University of Wolverhampton, School of Computing. May 1993.

Davies, W. S. (1983) *Systems Analysis and Design: A structured Approach.* Addison Wesley.

Dearnley, P.A. Mayhew, P.J. (1983) In Favour of Systems Prototypes and their Integration into the Systems Development Cycle. *The Computer Journal.* Vol 26, No 1.

DeMarco, T. (1979). *Structured System Analysis and System Specification.* Yourdon Press.

Dickerson, K. R. Benton, R. H. and Atyeo, M. A. (1988). Rapid prototyping tools and techniques. *British Telecom Technology Journal.* Vol 6, No 4.

DSDM (1995a) *DSDM Manual.* First Edition Tesseract Publishing: Surrey UK. February 1995.

DSDM (1995b) *DSDM Manual.* Second Edition Tesseract Publishing: Surrey UK. December 1995.

DSDM (1999) *DSDM.* http://www.dsdm.org/index.htm (23rd Feb 1999).

Downs, E. Clare, P. Coe, I. (1988) *Structured systems analysis and design: application and context.* Prentice Hall.

Eaves, D. (1991) Varieties of prototypes. Proceedings of the 2nd Australian Information Systems Conference. Sydney, Australia, February 1991.

Eva, M. (1992) *SSADM Version 4: A users guide.* McGraw-Hill.

Floyd, C. (1984) A systematic look at prototyping. In **Budde, R, Kuhlenkamp, K K, Mathiassen, L and Zullighoven, H (eds)** *Approaches to prototyping.* Springer-Verlag, Berlin, FRG.

Friel, G. Budgen, D. (1997) Design transformation and prototyping using multiple viewpointa. *Information and Software Technology.* Vol 39, No 2.

Frost, S. (1994) *OMT - Taking it further: Developing client server systems using OO technology.* Select Software Tools Cheltenham, UK.

Galleta, D. F. Hufnagel, E. M. (1992) A Model of End-User Computing policy: Context, Process, Content and Compliance. *Information and Management.* Vol 22, No 1.

Galliers, R. (1992) Information Systems Research. Issues, methods and practical guidelines. Alfred Waller Information Systems Series.

Galliers, R.D. Land, F.F. (1987) Choosing Appropriate Information Systems Research Methodologies. *Communications of the ACM,* Vol 30, No 11.

Gehani, N. McGettrick, A.D. (1986). *Software Specification Techniques.* Addison-Wesley.

Gilb, T. (1988). *Principles of Software Engineering Management.* Addison-Wesley.

Gill, P. J. (1992) RAD Tools, techniques take graphic direction. *Software Magazine.* Vol 12, No 5.

Glass, R. L. (1977). *The Universal Elixir and Other Projects Which Failed.* Computing Trends.

Goodwin, C. (1993) Rapid Reaction. *Computing.* January 21st 1993.

Gougen, J. A and Tardo, J. J. (1986) An Introduction to OBJ: A Language for Writing and Testing Formal Algebraic Program Specifications. IN N. Gehani and A.D. McGettrick. *Software Specification Techniques.* Addison-Wesley.

Gould, J. D. and Lewis, C. (1985) Designing for usability: Key principles and what designers think. *Communications of the ACM.* Vol 28, No 3.

Graham, D. R. (1989) Incremental development: Review of nonmonolithic life-cycle development models. *Information and Software Technology.* Vol 31, No1.

Graham, D. R (1992) Incremental development and delivery for large software systems. *Proc IEE Colloquium on Software prototyping and Evolutionary Development.* London, UK.

Graham, I. (1991) Structured prototyping for requirements specification in expert systems and conventional IT projects. *Computing and Control engineering Journal.* Vol 2, No 2.

Gray, P. D. Kilgour, A. C. Wood, C. A. (1988) Dynamic reconfigurability for fast prototyping of user interfaces. *Software Engineering Journal.* Vol 3, No 6.

Green-Armytage, J. (1995) Chart. *Computer Weekly*, August 10th 1995.

Grottola, M.G. (1989) Preparing a formal prototype session. *Mini-Micro Systems.* January 1989.

Guitierrez, O. (1989) Prototyping techniques for different problem contexts. *SIGHI, special issue.Conference on Human Factors in Computer Systems (CHI 1989).* Austin Texas, USA.

Hall, J. (1995) Introduction to SSADM 4+. *Data Management 1995,* Worcester College Oxford. UK. April 6th 1995.

Hardgrave, B. C. (1995) When to prototype: Decision variables used in industry. *Information and Software Technology.* Vol 37, No 2.

Hardgrave, B. C. (1996) Prototyping and System success: An Empirical analysis of Association. *Journal of Computer information Systems*, Vol 36, No 2.

Hardgrave, B. C., E. R. Doke., N. E. Swanson. (1993) Prototyping Effects On The System Development Life Cycle: An Empirical Study. *Journal of Computer Information Systems.* Spring 1993.

Hardgrave, B. C. Wilson, R. L. (1994) An investigation of guidelines for selecting a prototyping strategy. *Journal of Systems Management.* April 1994.

Harker, S. (1988). The Use of Prototyping and Simulation in the Development of Large-Scale Applications. *The Computer Journal.* Vol 31, No 5.

Harris, J. R. Parker, D. W. (1987) Evaluation of rapid prototyping methodology in a human interface. *Proceedings of Interact 1987. 2nd IFIP Conference on Human Computer Interaction.* Stuttgart, Germany.

Hart, A. (1985) Knowledge elicitation: issues and methods. *Computer-aided design.* Volume 17, No 9.

Hartson, H. R. Smith, E. C. (1991) Rapid prototyping in human-computer interface development. Interacting with Computers. Vol 3, No 1.

Harwood, K. (1987) On prototyping and the role of the software engineer. *ACM SIGSOFT, software engineering notes.* Volume 12, No 4.

Haughton, H. (1989) Developing Communication Protocols- A prototypical approach. IN *Proceedings of the Seminar Series On New Directions In Software Development.Rapid Prototyping for the Software Developer.* Wolverhampton Polytechnic.

Hayball, C. Barlow, D. (1988) Skills support in the ICL Kidsgrove Bonding Shop- A case study in the application of the KADS Methodology. *Human and Organisational Issues of Expert Systems:* Conference proceedings: Stratford. UK.

Hekmatpour, S and Ince, D. (1986) Rapid software prototyping. *Open University technical report 86/4* Open University, UK.

Hekmatpour, S. (1987) Experience with evolutionary prototyping in a large software Project. *ACM SIGSOFT Software Engineering Notes.* Vol 12, No 1.

Hemsley, D. (1983). Committee for the Promotion of Electroplating. Electroplating. *A guide for designers and engineers.* (David Hemsley, Chairman). Prepared for the CPE by Joseph Edwards.

Hilal, D. K. Soltan, H. (1991) To prototype or not to prototype? That is the question. *Software Engineering Journal.* November 1992.

Horngren, C. T. and Foster, G. (1987). *Cost accounting. A Managerial Emphasis.* 6th Edition. Prentice Hall International Editions.

Hundy, B. (1991). Using Costs sensibly. *Metals and Materials.* Volume 7, No 2.

Iivari, J. and Karjalainen, M. (1989). Impact of Prototyping on User Information Satisfaction During the IS Specification Phase. *Information and Management.* Vol 17. No 1.

Ince, D. (1988) *Fashioning the Baroque.* Oxford University Press.

Ince, D. (1991) The making of a modern methodology. *Informatics.* Vol 12, No 6.

Ince, D. (1992) Keynote Address. *Proceedings of the IEE Colloquium on Software Prototyping and Evolutionary Development* London. November 11th 1992.

Jackson, M. (1983) *System development.* Prentice-Hall.

Janson, M. A. Hammerschimidt, J. P. (1990) Managing the information systems development process: the case for prototype and pilot systems. *International Journal of Information Management Science (Taiwan).* Vol 1, No 1.

Jarvenpaa, S. (1988) The importance of Laboratory Experimentation in IS Research. *Communications of the ACM.* Vol 31. No 12.

Johnson, H.T. (1989) Managing Costs: An outmoded philosophy. *Manufacturing Engineer.* Vol 102, No 6.

Johnson-Laird, P. N. (1983) *Mental Models.* Cambridge University Press.

Jordan, P. W., Keller, K. S., Tucker, R. W., Vogel, D. (1989) Software storming: combining rapid prototyping and knowledge engineering, *Computer,* May 1989.

Jorgensen, A. H. (1984) *On the psycology of prototyping.* in **Budde, R, Kuhlenkamp, K K, Mathiassen, L and Zullighoven, H (eds)** *Approaches to prototyping* Springer-Verlag, Berlin, FRG.

Karten, N. (1991) Integrating IS Disciplines into End User Training. *Journal of Management Information Systems.* Vol 8, No 1.

Kautz, K. McMaster, T. (1993) Introducing Structured Methods: An Undelivered Promise? - A Case Study. *Scandinavian Journal of Information Systems.* Vol 6, No 2.

King, J. (1990) 4GLs: INTRO. *Infomatics.* October 1990.

Kinmond, R. M. (1994) *'Prototyping: three perspectives'* MSc Thesis University of Wolverhampton.

Kinmond, R. M. Stephens, M. A. (1995) *Current Trends in Software Development: An analysis of commercial prototyping practice.* Technical report. University of Wolverhampton.

Kirk, J. and Miller, M.L. (1986) Reliability and validity in qualitative research. Sage.

Klausner, A. and Konchan, T. E. (1986) Rapid Prototyping and Requirements Specification using PDS. *Software Specification Techniques.* Edited by N. Gehani and A.D. McGettrick. Addison-Wesley.

Klein, L. and Newman, W. (1987) A strategy for integrating human-computer interface considerations into Alvey-2 application projects. *Alvey Human Interface Club.* September 1987.

KPMG (1990) *Runaway Computer Systems.* KPMG Peat Marwick.

Kull, D (1986) Anatomy of a 4GL disaster. *Computer Decisions* Vol 18.

Lamb, D. A. (1988) *Software Engineering: Planning for Change.* Prentice Hall. Englewood Cliffs.

Lano, K. Haughton, H. (1994) *Object-oriented specification case studies* / edited by K. Lano and H. Haught. Prentice-Hall.

Lee, S., Yen, D., Lee, S.M. and Tang, H.L. (1991) Impact of End Users Direct Access to Corporate Databases on End Users' Decision Making and Programming Activity: An Exploratory Study. *Journal of Computer Information Systems.* Vol 31, No 3.

Lehman, M. M. (1980) On Understanding Laws, Evolution and Conversation in the Large-Program Life Cycle. *Journal of Systems and Software.* Vol 1, No 3.

Lehman, M. M. (1988) Software - Promise and Threat. (1989) *Proceedings of the Shell conference. Logistics: Where ends have to meet.* Appledorn, Netherlands, November 1988.

Lehman, M. M. (1991) Software engineering, the software process and their support. *Software Engineering Journal.* Vol 6, No 5.

Lichter, H. Schneider-Hufschmidt, M. and Zullighoven, H. (1993) Prototyping in industrial software projects-Bridging the gap between theory and practice. *IEEE Transactions on software Engineering.* Vol 20, No 11.

Lientz, B. P. Swanson, E. B. (1980) *Software Maintenance management.* Reading MA: Addison-Wesley.

Lin, Chang-Yang (1990) Systems Development with Application Generators: An End User Perspective. *Journal of Systems Management.* Vol 41, April 1990.

Linkman, S. G. Walker, J. G. (1991) Controlling programmes through measurement. *Information and Software Technology.* Vol 33, No 1.

Livesey, P. B. (1984) *Experience with prototyping in a multi national organisation,* **In Budde, R, Kuhlenkamp, K K, Mathiassen, L and Zullighoven, H (eds)** *Approaches to prototyping* Springer-Verlag, Berlin, FRG (1984)

Londeix, B. (1987) *Cost Estimation for Software Development.* Addison-Wesley.

Low, G. C. Jeffery, D. R. (1990) Function Points in the estimation and evaluation of the software process. *IEEE Transactions on Software Engineering.* Vol 16, No 1.

Luqi, L. (1989) Software evolution through rapid prototyping. *IEEE Computer* Vol 22, No 5.

Luqi, L. (1991) The role of prototyping languages in CASE. *International Journal of Software Engineering and Knowledge Engineering.* Vol 1, No 2.

Luqi, L. Berzins, V., Yeh, R. (1988) A Prototyping Language for Real-Time Software. *IEEE Transactions on Software Engineering.* October 1988.

Luqi, L. Cooke, D. (1992) The Management of Uncertainty in Software Development. *IEEE 16th International Computer Applications Conference Conference.* Chicago, USA, September 1992.

Luqi, L. Steigerwald, R. (1992) Rapid software prototyping. *Proceedings of the 25th Hawaii International Conference on Systems Science.* Hawaii. January 1992.

Madhavji, N. H. (1991) The process cycle. *Software Engineering Journal.* Vol 6, No 5.

Maglitta, J. Hildebrand, C. (1993) A Bridge too late. *Computerworld,* May 31st 1993.

Martin, J. (1986) *Information Engineering.* Savant research studies, UK.

Martin, J. (1991) *Rapid Application Development.* Macmillan Publishing, New York, 1991.

Martin, M. P. Carey, J. M. (1991) Converting prototypes to operational systems: evidence from a preliminary industrial study. *Information and Software Technology* Vol 33, No 5.

Mason, R. E. A. Carey, T. T. (1983) Prototyping Interactive Information Systems. *Communications of the ACM.* Vol 26, No 5.

Mathiassen, L. (1984) *Summary of the working group.*In **Budde, R, Kuhlenkamp, K K, Mathiassen, L and Zullighoven, H (eds)** *Approaches to prototyping* Springer-Verlag, Berlin, FRG (1984).

Mathiassen, L. Seewaldt, T. Stage, J. (1995) Prototyping and Specifying: Principles and practices of a Mixed Approach. *Scandinavian Journal of Information Systems.* Vol 7, No 1.

Mayhew, P. J. (1987) *An Investigation of Information Systems Prototyping.* PhD Thesis. University of East Anglia 1987.

Mayhew, P. J. (1990) Software prototyping: implications for the people involved in systems development. *2nd Nordic Conference on advanced information systems engineering.* Stockholm, Sweden, May 1990.

Mayhew, P. J. Dearnley, P. A. (1990) Organization and management of systems prototyping. *Information and Software Technology.* Vol 32, No 4.

Mayhew, P. J, Worsley, C. Dearnley, P.A. (May 1989) 'Control of Software Prototyping Process: Change Classification Approach' *Information and Software Technology* Vol 31 No 2.

Mayhew, P. J. Worsley, C. (1992) Software Prototyping: The Management Implications. *Proceedings of the IEE Colloquium on Software Prototyping and Evolutionary Development* London. November 11th 1992.

Mclean, E. R. Kappelman, L. A. (1992) The Convergence of Organisational and End-User Computing. *Proceedings of the 25th Hawaii International Conference on Systems Science.* Hawaii. January 1992.

Metal Finishing Association. (1973) *The Uniform Costing Scheme of the Metal Finishing Association.* The Metal Finishing Association.

Morrison, W. (1988) Communicating with users during systems development. *Information and Software Technology.* Vol 30, No 5.

Naumann, J. D, Jenkins, A. M (1982) Prototyping: The new paradigm for systems development. *Management Information Systems Quarterly.* Vol 6, No 3.

NCC 1988. *Introductory Guide.* The Analysis and design Sampler Pack. Edited by S.Platts. NCC Ltd.

Necco, C R, Tsai, N, and Gordon, C L (1989) Prototyping: Use in the development of computer based information systems. *The Journal of Computer Systems.* Fall 1989.

Ould, M.A. Roberts, C. (1987) Modelling iteration in the software process. Proceedings of the 3rd International Software Process workshop Colorado, U.S.A. November 1986.

Palvia, P, Nosek, J.T (1990) An empirical evaluation of system development methodologies. *Information Resources Management Journal* Vol 3, No 3.

Parbst, F. (1984) Experience with prototyping in an IBM based installation in **Budde, R, Kuhlenkamp, K K, Mathiassen, L and Zullighoven, H (eds)** *Approaches to prototyping* Springer-Verlag, Berlin, FRG .

Parks, M. W. (1987) Time, Cost Reduced By Developing Manufacturing Information Systems Using Microcomputer Prototyping. *Industrial Engineering.* Vol 19, No 6.

Pilote, M. (1988). AI Prototyping-Can it be Integrated in Business? *The Proceedings of the Fourth Annual Artificial Intelligence and Advanced Computer technology Conference.* Issued by Tower Conference management. Long Beach California USA. May 1988.

Pliskin, N. and Shoval, P. (1989). Responsibility sharing between sophisticated users and professionals in structured prototyping. *Information and Software Technology.* Volume 31, No 8.

Pressman, R. (1992) *Software Engineering: A practitioners approach.* McGraw-Hill International edition.

Prizant, A. (1986) Is prototyping counterproductive? *Information and Software Technology.* Vol 28, No 7.

Purtilo, J. Larson, A. Clark, J. (1991) A methodology for prototyping-in-the-large. *13th International Conference on Software Engineering* Austin, Texas, U.S.A. May 1991.

Ralston, D. W. (1988) *Principles of Artificial Intelligence and Expert Systems Development.* McGraw-Hill Book Company.

Redmill, F. J. (1989) Computer system development: problems experiencedin the use of incremental delivery. *Conference on Safety of Computer Control Systems* Vienna, Austria, December 1989.

Rittenberg, L. E. Senn, A. (1993) Is it True that End-User Computing is like a Runaway Train? *Internal Auditor*, February 1993, Vol 50, No 1.

Rockart, J. Flannery, L. (1983) The Management of End User Computing. The Communications of the ACM. Vol 26, No 10.

Royce, W.W. (1970). *Managing the Development of Large Software Systems: Concepts and Techniques.* Proceedings WESCON.

Rzevski, G. (1984) *Prototyping versus pilot systems: strategies for evolutionary information systems development.* in **Budde, R, Kuhlenkamp, K K, Mathiassen, L and Zullighoven, H (eds)** *Approaches to prototyping* Springer-Verlag, Berlin, FRG.

Rumbaugh, J. Blaha, M. Premerlani, W. Eddy, F. Lorensen, W. (1991) Object-oriented modeling and design. Prentice Hall International.

Scott Gordon, V . Bieman, J. M. (1994) Rapid Prototyping: Lessons Learned. *IEEE Software,* Vol 12, No 1.

Sheskin, D.J. (1997) Handbook of Parametric and Nonparametric statistical functions. CRC Press.

Shoval, P. and Pliskin, N. (1988). Structured prototyping: integrating prototyping into structured system development. *Information and Man Management (Netherlands)*. Vol 14, No 4.

Smith, M.F. (1991) *'Software Prototyping Adoption, Practice and Management'* McGraw-Hill, London.

Sommerville, I. (1992) *Software Engineering.* 4th edition. Addison Wesley.

Sommerville, I. (1995) *Software Engineering.* 5th edition. Addison Wesley.

Stephens, M. A. Bates, P. E. (1990) Requirements engineering in the development of an estimating system. *Information Software Technology.* Vol 32, No 4.

Stephens, M. A. Bates, P. E. (1993) Controlling Prototyping and Evolutionary Development. *IEEE 4th International Workshop on Rapid System Prototyping.* Research Triangle Park. North Carolina USA. June 1993.

Symons, C. R. (1988) Function Point Analysis: Difficulties and Improvements. *IEEE transactions on software Engineering.* Vol 14, No 1.

Symons, C. R. (1991) *Software Sizing and Estimating: Mk II FPA.* John Wiley, Professional Computing Series.

Tanik, M. M. Yeh, R. T. (1989) Rapid Prototyping in Software Development. *Computer.* Vol 22, No 5.

Tate, G. (1990) Prototyping: helping to build the right software. *Information and Software Technology.* Vol 32, No 4.

Thomas, M. (1987) *Structured Techniques for the Development of Expert Systems.* BCS Conference.

Thomas, M. (1990) see pers comm (p54)

Tozer, J. E. (1987) Prototyping as a system development methodology: opportunities and pitfalls. *Information and software technology* Vol 29, No 5.

Tully, C.J. (1987) Software Process Models and Iteration. *Proceedings of the 3rd International Software Process Workshop.* Colorado, U.S.A. November 1986.

Veraart, V. E. Wright, S. L. (1995) Software Engineering Education - Adding Process to Projects: Theory, Practice and Experience. *Proccedings of the 1995 Asia pacific Software Engineering Conference.* Brisbane, Australia. December 1995.

Vienneau, R. (1997) A Review of Formal Methods. In: Dorfman, M. and Thayer, R.H. (Eds) *Software Engineering.* IEEE Computer Society Press. Washington, USA.

Vonk, R. (1990) *Prototyping: - The effective use of CASE Technology.* Prentice-Hall.

Walker, C. (1995) Sainsbury shelves support strategy after staff survey. *Computer Weekly.* 13th April 1995.

Warhurst, R. Flynn, D. (1990) Validating JSD specifications by executing them. *Information and Software Technology.* Vol 32, No 9.

Weber, R. (1986) Planning and control issues in end-user computing. *The Australian Computer Journal.* Vol 18, No 4.

Weiner, M. (1986a) Good Days and Bad Days in the Finishing shop. Cost and Pricing Part W-Pricing Methods Party. *Metal Finishing.* Vol 84, No 1.

Weiner, M. (1986b) Good Days and Bad Days in the Finishing shop. Cost and Pricing Part X-Sliding Scale. *Metal finishing.* Vol 84, No 2.

Welch, O. J., Gambill, S., Ramparu, N., Parzinger, M. (1992) An analysis of Techniques Employed to Measure the Effectiveness and Productivity of End User Computing. *IRMA Conference proceedings.* Idea group publishing.

Wells, T.D. (1987) On the conflict between development models and development methodologies. *Proceedings of the 12th structured methods conference.* Chicago, U.S.A. August 1987.

Whitehead, G. and Upson, A. (1982). *Success in Accounting and Costing.* John Murray.

Wild, R. (1985). *Essentials of Production and Operations Management.* 2nd Edition. Holt Reinhart and Winston.

Williams, K. (1992) *Software Prototyping Theory and Applicationn:* BSc Project Report. School of Computing, Wolverhampton Polytechnic. May 1992.

Yourdon, E. (1989) *Modern Structured Analysis.* Prentice-Hall International. 1989.

Appendices

Appendix 1

Background to developing an information system for a metal finishing company

Appendix 1

Background to developing an information system for a metal finishing company.

Frost Electroplating, a medium sized metal finishing company, identified the need for a formal method of estimating and pricing. The company had long established informal methods which were felt to be in need of improvement.

It was required to carry out the following processes to investigate their problems:

- To analyse the current practices in estimating and pricing, and report the cause of the existing problems.

- To investigate possible solutions using appropriate computer techniques.

- To build a system to reduce the incidence of erroneous and inconsistent prices with particular regard to possible future industry changes and maintenance needs.

Prior to the research being carried out, the requirements for a pricing and estimating system were discussed with the company's normal software suppliers. Although Frost Electroplating had a comparatively long history of computerisation (particularly for a small company) and their suppliers had written a large quantity of software for a wide variety of different tasks, this particular problem was considered to be "too difficult". In addition a review of current commercially available computerised solutions in the area did not reveal a suitable system.

The project started by attempting to investigate the problem using conventional systems analysis methods. These were discovered to be unsuitable. The research then concentrated on investigating how software prototyping can be useful to the user and developer and how it can be integrated into the overall system design process. This approach was investigated extensively during the project.

The problem at the company

In an ideal commercial business environment a selling price should be based on an accurate estimate of likely costs with a view to making a profit on each contract that is tendered for. In reality this is not the case!

There are considered to be particular problems within the metal finishing industry and this difficulty is acknowledged by the Committee for the Promotion of Electroplating(CPE) (Hemsley 1983) who state that:

> "It is impossible to give firm guidelines on prices."

The research set out to address the problems with a view to producing a practical system which could be used as part of a solution to the problem of pricing and estimating within the metal finishing industry.

It was anticipated that due to the difficulty of the problem, it would be essential that the method of producing the system and the way in which it was presented should aid developer/user communication at all times. Therefore the emphasis throughout the research was on producing a system which could be understood and verified by the appropriate company personnel.

Definition of the business terms used in the research

The terms estimating, pricing, and costing required careful definition before an attempt could be made to investigate the problems associated with them. Within the context of this research project, the following definitions have been used:

- Costing: The determination and analysis of cost, where cost is the expense incurred in producing a product.

- Estimating: The process of calculating the likely costs involved in producing a batch of components plated to a particular finish, prior to the production taking place.

- Pricing: The calculation of a figure at which a product is intended to be sold, normally with the intention of making a profit. This need not be necessarily directly related to costs.

The definitions are based on the terms used within the company. The relationship between the three terms is illustrated in figure A.1.

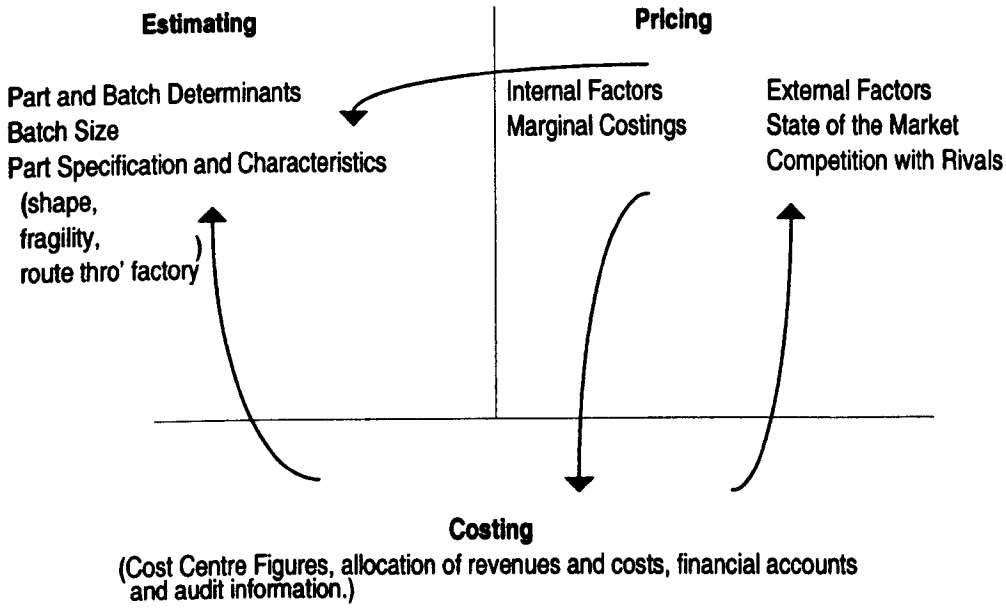


Figure A.1. The relationships between costing, pricing and estimating

The problems of costing, pricing and estimating in the metal finishing industry

Within the metal finishing industry a formal method of estimating and pricing based on individual job costs is very difficult to operate. The Uniform Costing Scheme of the Metal Finishing Association (MFA 1973) states that:

> "The cost of each job is either unascertainable or can seldom be obtained at an economical cost".

Companies in the industry do not generally produce a constant output of uniform batches of components with easily identifiable costs associated with their production. Jobs for tender come in a diverse range of shapes, size, batch quantity, and required finish specification. Often the batch will be a one-off and as a result the price quoted can only be on the basis of experience with previous work rather than the known production costs of the batch under consideration. An estimator (the person in the company responsible for formulating prices and sending out quotations) must make a pricing decision based on intuitive knowledge, as he or she will often not have an accurate breakdown of the costs involved when making that decision. The nature of the industry is such that there are almost as many pricing methods as there are finishing shops. (Weiner 1986a).

In manufacturing industry there are examples where costing, can be carried out in a more formal manner according to well documented techniques. Job costing (Horngren

and Foster 1987) and process costing (Wild 1985) are examples of costing techniques that can be used to suit particular applications. If this type of analysis is available then various cost based formulas are available for guiding the pricing decisions. However the pricing decisions will be influenced by the method of allocating costs as well as the actual costs themselves; and as there are various acceptable methods it is difficult to argue that pricing can be based exclusively on costs (Bull 1984).

Engineering literature often addresses the problem of controlling costs and the nature of the cost accounting process, (Brimson 1988, Hundy 1991, Johnson 1989). Unfortunately the trend is for discussion of the problem of costing at an abstract level. There is very little information on tackling the way that a practical costing, estimating, or pricing system can be developed and run by a metal plating company.

Smaller plating companies offering a limited variety of finishes and who deal with fixed batch quantities and a small product range are more likely to be able to give fairly accurate estimates of job costs, as the levels of complexity of process and component type will be less. Hence the price charged can more accurately reflect the estimate of the costs involved.

As the range of finishes offered, the processes to apply them and the number of different types of component expand, the complexity of the task of estimating and pricing greatly increases.

An example of the processes required to plate a comparatively straightforward finish is given below.

The Finishing Handbook and Directory (1988) states:

"A typical treatment sequence for the nickel plating of steel articles is as follows:

After any necessary mechanical cleaning:

1. Preclean to remove heavy soiling.
2. First stage electrolytic alkaline clean (cathodic or anodic).
3. Water rinse.
4. Dilute hydrochloric acid dip if descaling or rust removal required.
5. Water Rinse.
6. Second stage electrolytic cleaning.

7. Water Rinse.
8. Dilute acid dip for surface activation.
9. Water Rinse.
10. Nickel Plate.
11. Rinse.
12. Dry. (If no further treatment is required)."

Complex jobs with more finishing required will need a greater number of processes to achieve the final finish. The more complex the work entailed, the higher the probability that the actual costs incurred will vary from any estimated costs. In addition, within a company there may be a variety of production processes which carry out the same task (i.e. apply the same finish) but will be used on different part types due to the particular physical properties of that part (e.g. weight, fragility, or quantity in a batch).

Hence a pricing structure based on job cost analysis still requires accurate estimation of the likely job costs by the estimating staff and will not itself create the answer to the problems of pricing and estimating. This takes considerable skill to do accurately. The final part of a good pricing framework should be the ability to state that the overall profit of the company is generated by profit made on **each** job, (Weiner 1986b). An overall level of profit without this confidence, suggests a less than optimum pricing structure.

The CPE give an indication of the basis on which prices can be calculated from a company's overall costs. However the figures given are very general and are based on a "typical" plating company so cannot be truly representative of each firm operating in the field.

The problems are further complicated by the external considerations of the market. It is possible for a company to set a particular price by beginning with some knowledge of the costs of production and adding a profit figure related to the profits of the company (Whitehead and Upson 1982); however this is complicated by the acknowledgement that the final price may not be related to cost, but to what the company thinks the customer is prepared to pay.

It can be seen that pricing in a way that will allow a company to consistently be aware of the level of profitability that it is generating is a problem of considerable complexity.

The particular problem at Frost Electroplating

The company carried out a very wide variety of metal finishing processes. These were mainly, plating of metal components using a combination of different electroplating techniques. However other forms of metal finishing were also used e.g. enamelling, lacquering, and various other forms of finish. The total range of finishes was very extensive and ranged from basic single process jobs, to complex multi-plate processes involving multiple pre-plate and post plate operations.

Because of the very wide range of processes available and the great number of different types of component sent in for tender the problems of pricing at Frost Electroplating were extensive and complex. The policy of the directors was to continue to offer a wide range of finishes in order to maintain a large customer base although this does make the pricing process more difficult.

By offering such a comprehensive service the company encountered serious problems with the estimating and pricing of work. This was exacerbated by the large number of components that required estimates.

It was felt that the existing methods of estimating and pricing were not sufficiently analytical or consistent to allow the best prices to be produced. The identified problems included:

- The relationship between cost and price was unclear.
- A price quoted may be too low and the company may not show a profit on the transaction.
- A price quoted may be too high in which case the tender is not likely to be accepted by the customer and a competitor will gain the contract.
- Different prices may be quoted for similar work which can lead to confusion for customers and embarrassment at the company.

In isolation or combined together it was felt that an attempt should be made to minimise or to solve these problems.

Initial problem analysis

The research started by investigating the current pricing and estimating procedures at the company. It was felt that although the problem under investigation was difficult, with a lack of precise predetermined requirements, it would be sensible to start from a position that is used in many commercial computing applications: i.e. a traditional specifying approach. The use of an ad-hoc development model was rejected as being of little value to the exercise.

By using a structured design method it was hoped to build up a picture of existing practices within estimating and pricing. This could be used as a starting point to analyse the current problems and develop a specification for a solution. After considering the various methodologies a method based on the Yourdon model was chosen, mainly because of the expertise that existed within the computing department at Wolverhampton.

Additionally, most of the literature available at the time did not make clear how a software project developed using prototyping of any form would commence. So an investigation of current practices seemed an appropriate place to start.

The initial stage of investigating the current estimating and pricing practices was commenced by interviewing and observing the estimating and pricing staff at work. Notes were taken on the procedures that they adopted and the interview sessions were documented.

At the same time, members of the company's senior management and various directors were interviewed to try to discover what influences they had in the estimating and pricing process. The intention was to construct a model of the current system using data flow diagrams derived from the interviews and analysis of procedures. This forms the first part of the Yourdon analysis method (Yourdon 1989). The current system is modelled and the essential parts of it are derived which show how it meets its stated purpose.

After this, the existing essential parts of the current system are combined with the new requirements to form a model that describes the functions of the new system. Eventually after an eight stage process the Yourdon design method should enable the modules to be designed that will carry out the tasks specified. Finally implementation

on the company's computer hardware using a suitable programming language would be carried out.

The intention was to build a specification using the sequential stages as suggested by the design method.

Results of the initial study

The following results were obtained after meetings with estimators, and members of the management and directors of the company.

The pricing and estimating system in use at Frost Electroplating was not based on well defined procedures which could be easily documented. The factory had commercial and business principles which formed the basis for the overall estimating and pricing system. Analysis of prevailing market conditions combined with an interpretation of the company's cost data was used to form the pricing policy. Bull (1984) states that when companies set a selling price:

> "It is often argued that full cost must be known before the selling price can be fixed. This concept is most attractive at first sight, yet for many industries price is fixed by considerations external to the firm.

At Frosts the management were aware that external pressures were an extremely significant part of the pricing process and that costs alone will not dictate price.

The estimating and pricing was not carried out using the same techniques across the company. Different finishes were priced using different methods. This had developed as a result of the internal organisation of the factory.

The staff operating the pricing system did not have a continuous source of information from which to amend and update prices and as a result some parts of the pricing scheme were more independent of analysis of costs than might have been considered desirable.

Summary of specific problems highlighted during the investigation

The following sections briefly outline the problems that occurred during the initial analysis stage.

Time and complexity in producing prices for customers

At Frosts the estimating and pricing was carried out by several different staff within the company. The time taken to research, create, and prepare each individual quotation was substantial. There were several reasons for this:

- The prices on quotations were checked against previous records to ensure a level of consistency in the pricing policy. Quotation records were filed in date order. Searching for several related quotations could take a great deal of time, whilst there was no certainty that all the records have been accessed.

- The production and raw material costs involved when creating a quotation were often likely to have changed. When comparing two quotes a calculation to align these different costs was likely to be required.

- Many quotations were checked by at least one other person as well as the individual who prepared it.

- Quotations for all non-precious metal plating were written out in long hand. Alterations could require the production of a new document. Both of these processes were very time consuming.

- Unless the job was very straightforward (e.g. a single order for a single process operation for which pricing tables existed) the calculation of a price was done manually. For orders involving several different processes in different areas in the factory these calculations could be laborious.

Diversity of components and finishes available

The nature of the plating business tends to require a wide possible range of finishes to be applied to an ever increasing number of types of component of varying size, shape, and substrate. As a result the allocation of a standard known cost on which a selling price can be based is often not possible. Although it is likely that a similar job might have been priced before, it is rare for an identical set of finishes on an identical component to be requested. Price tables for simple finishes on components of a set size existed but only covered a very limited part of the estimating and pricing domain. As a result a new price had to be calculated by the estimators for each request as it arrived.

Different approaches to calculating prices

The subjective nature of the pricing process lead to inconsistencies in pricing. Although prices were checked against previous quotations it was impossible to guarantee consistency. This was unfortunately inevitable when several different staff used their own acquired techniques when they are carrying out their work. There were only very limited formal written pricing procedures. Most of the pricing at Frosts relied heavily on the individual skills of experts. Where information was not available at the time of the estimate being prepared, or if particular trading conditions have caused the usual practices to be amended, disagreements over particular pricing decisions occurred.

Difficulties in articulating techniques

The reluctance or inability of certain personnel to explain the way that they came to their final decisions on estimating and pricing also made it difficult to obtain specific information about pricing policy. In addition although estimators could help with specific examples, trying to develop these into basic rules was not possible.

By using structured analysis some useful insights into the problem were discovered but it was difficult to decompose parts of the remaining problems into more detailed sections. The estimators although willing to describe the way that they arrived at prices found it very difficult to give any general rules that could accurately represent their decision making processes. The very large number of individual examples that were cited as "special cases" made analysis of underlying procedures extremely difficult to carry out. This problem is not unfamiliar and is a recognised problem when trying to elicit information from experts with specialist knowledge. Hart (1985) states:

> "In short, the expert can often tell you the decision, i.e. 'what', but not describe the process, i.e. the details of 'how'."

It is recognised within the literature (E.g. Cornes 1990, Bansley and Bødker 1993, Kautz and McMaster 1994) that there are particular problems with structured analysis such that it does not always provide the most appropriate means for analysing the problem.

No standards for quotation documentation

Because the problem area was complex and written pricing procedures did not exist, quotations produced did not conform to an easily interpretable standard form.

Costing information not always available

The methods of costing in use at Frosts did not give costs of individual jobs carried out in the factory. It was therefore difficult to predict accurately that a price quoted for a job would produce a profit when it has been completed and paid for. Only by analysing longer term financial accounts was it possible to recognise performance trends within individual cost centres, after which prices could be modified accordingly.

Information availability was a key factor for accurate quotation production; the more information available then the better prepared a quotation would be. This leads to well, researched prices and a high rate of acceptance of the price by the customer. Conversely, lack of information could lead to loss making work being carried out, or a high level of quotation price rejection.

Conflicting aims of different users

During this study various groups of users (management, production staff, estimating staff) were required to participate in the process. There was difficulty in reconciling the aims of the parties. This reflects the experience of Buckley *et al* (1988) who discovered different priorities between end-user needs and management requirements.

Conclusion of initial investigation

The factors that formed the pricing structure were combined and weighted by decisions taken by management in response to constantly changing operating conditions.

It became clear during the first stage that the rules were far from comprehensive and required large numbers of exceptions to be made to them. The relationship between costs and prices remained unclear and the transformation of determinants into quotations could still not be determined. To improve this, a more formal relationship between costing and pricing needed to be developed.

This work formed a basis for discussion with management and helped identification and prioritisation of the main problems:

- Quotations were inconsistent (This was considered to be the most significant problem)
- Cross referencing of previous quotations was too difficult due to insufficient indexing and the lack of a standard layout
- Manual quotations were expensive to produce and modify

Following these discussions, the management, who were now more aware of the extent of the problems, wished to reformulate the requirements of the proposed system. However, they were still not fully aware of what they actually wanted. This phenomenon is recognised by Yourdon (1989) who states that one of the good candidates for prototyping is where:

> "The user is unable to articulate (or "prespecify") his or her requirements in any form and can only determine the requirements through a process of trial and error."

It was concluded that pricing was a function of:

- Type and specification of Finish.
- Processes to apply finish.
- Attributes of the part.
- Customer type.
- Nature of quotation enquiry.
- Factory loading.
- Competition.

However the precise inter-relationships still remained unclear.

The use of the Yourdon analysis technique should be regarded as a partial success. There is no doubt that the developer learnt a great deal about the system requirement. The use of various diagrams and documents did clarify some of the issues. However it was felt that a less abstract approach was required where users could actually see parts of a potential system would be valuable, particularly where ambiguity and uncertainty still remained.

As a result of this investigation it was decided that further analysis using conventional methods would probably waste time and that little extra information would be discovered. Hence it was felt that techniques that allowed better feedback from users would give the development team a better insight into the way that the specification should be developed. It was therefore decided to adopt prototyping as the main method to continue with the project.

Appendix 2a

Outline system specification provided for experiment

Appendix 2a

Outline system specification provided for experiment.

Large Fabrications plc.

Large Fabrications plc are a software house building information systems for manufacturing and the service sector. They use Pascal, FoxPro, COBOL, ORACLE, C, C++ and various assembler languages. In addition they have a variety of different CASE software (Select, StP, Oracle CASE). The developments are for a wide range of different environments and include PC standalone (DOS and WINDOWS), PC networks, UNIX, and AS400 platforms.

After 25 years building software they have decided to try and formalise some of their procedures. They are aware that they use lots of environments, lots of tools, lots of different programmers, managers, styles of working. They want to find out how the different projects measure up in terms of quality and adherence to targets/budgets. They want to concentrate in the areas that they feel produces the best cost/benefit ratios. Despite the hype surrounding so-called high productivity tools they are not always convinced that the investment pays off. They want to record as many features as possible of their software development projects in order to try and improve the way that they do things.

In order to do this they need to build a suitable tool to help them, and as is always the case they have a reasonable idea of what is required but the specification is not complete!

However they think they have identified certain defined steps within the project which correspond well to functions that the tool must have implemented. Hence they feel it is possible to prioritise the different sections of the system. Recently they have heard a lot about *incremental prototyping* and how developing systems in increments allows a partial payback for a customer over a short time span. The senior management is beginning to feel that some of their developments are trying to build too much in one go and that it would be better to build software in smaller "bite sized pieces"! Obviously this is not possible for all software. (After all you cannot have an avionics system that supports a plane taking off but not landing!), but for business systems it might be better to go for this *incremental* approach since business needs and user requirements change so rapidly.

Hence they intend to use this approach for the development of the tool. The management have decided that they will **not** try to implement the whole system in one go. This *monolithic* method has lead to problems in the past with very considerable time and cost over runs.

On this occasion they are going to stick to subsections of the whole system which will be built according to strict time segments. These segments will be 4-6 weeks long with a review meeting at the end of the period. The review will be **mandatory**. At the review a decision will be taken as to how the particular segment under construction will proceed.

It must be stressed that the company wants to spread good practices throughout its workforce and improve the quality of its products. It feels that high *individual productivity* is not necessarily a good thing. This is because systems built at high speed may compromise quality which can lead to a loss of confidence by customers and any benefits in originally building systems very quickly are lost as maintenance problems build up.

Statistics about the details of a particular programmer are of no interest to the company, they believe that only by co-operation within teams can they become more successful. Hence the fact that a person wrote 500 lines of Pascal or FoxPro in one day is of little relevance especially if it doesn't work as the customer requires.

The senior management have decided to split the development into 2 increments. The first phase is to get a basic usable system up and running. The second phase will add extra features.

Business Objective of Implemented System.

To allow management to collate information about software projects and processes, to enable the software development processes to be improved.

This is a provisional list only and has been drawn up from very limited contact between a trainee analyst/programmer and a senior member of the company management.

It must be stressed that the requirement is very likely to change!

System Objectives. (Increment 1).

The personnel within the company are divided into management, administration, and staff. The managers have access to all information and to all functions. Administrators have the authority to add new projects and staff but do not have such extensive querying capabilities. Staff can log activities and can get information on their own work breakdown only. This means that the system must have three levels of access - hence password protection is required.

The features below are split into those that the managers require and those that other personnel will need to use.

1) Project Manager

Inputs.
Set up new project

Set up new staff member (Initials plus password)

Set up new activity (e.g. coding, documentation etc.)

Add software/hardware etc. to resource list.

Projects - PR + 4 numerics
E.g. PR0001 New order tracking system for Bilston Expanded Tubes plc

Staff Initials (up to 4)
E.g. MAS (a password will also be required to facilitate input into system. System will not allow duplicates)

Activity AC + 4 numerics
E.g. AC0002 Write Draft Specification

Outputs.

All of the following three reports to be on screen with option to print or export to txt file:

- List all people + time spent per project.
- List all projects with people, time, tools used.
- Resources used on project. (E.g. what CASE software, laser printer, SUNs etc.).

A manager must also be able to schedule their own activities as a normal team member. However an extended list of activities is available to them. (E.g. , plan budget, review project, play golf).

The manager can set up these activities as he/she would any other.

2) Team Member

Inputs.
The main build requires that team members should be able to enter the time that they have spent on various activities whilst at work.

Enter Initials

Enter Password

Data entry by week. and then by project. (Cannot enter data against non-existent project)

Inputs against an increment, or against total system, or against general duties, and what they were actually doing (I.e. coding, testing., design review, making the tea etc.).

Outputs.
Report showing activities for a given week or month. Team member can only access their own record **unless** they are a manager in which case access to any information is available. To be ordered by date.

3) Administrator

Inputs and Outputs

Same input options as a manager , same output options as a staff member.

Priority of Implemented Increments. (It is very important that these are adhered to.)

Increment 1 - First Stage

- Password
- Set up new project
- Set up new staff member
- Set up new activity (e.g. coding, documentation etc.), data entry.

Increment 1 - Second Stage

- List all people + time spent per project
- List all projects with people involved and time spent
- Resources used on project. (E.g. what CASE software, laser printer, SUNs etc.).

By the end of phase one, personnel will be able to log their time spent on particular projects and management will have a way of generating reports giving a breakdown of these statistics.

By the end of phase one all implemented functions will be represented in the design documentation. Failure to do this will result in project management penalties.

Appendix 2b

Summary data for field experiment

Table A2 - 1. Summary Data for Field Experiment

Table Showing Data on Dependent and Independent Variables

Ranked on Volume of Communication

Groups	Sales	Specification	Reliability	Std of Construction	Maintainability	*Completeness	Overall Quality	*Prototyped?*	*Communication*	Comments
		Scale 0-7	Scale 1-3	Scale 1-3	Scale 1-3	Scale 0-5	Scale 0-9			
M19	0	0	1	1	1	0	0	*No*	*0*	No implementation
T01	0	0	1	1	1	0	0	*No*	*0*	No implementation
M3	0	5	2	1.4	1.4	1	4.8	*No*	*0*	Complex
M4	0	5.5	2	1.66	2	2	5.66	*No*	*0*	Unreliable, buggy,has potential
M5	2	3.25	2.6	1	1	5	4.6	*No*	*0*	V.limited. Hard to maintain
T2	0	2	1	1	1.67	1	3.67	*No*	*1*	Poor spec interpret, unreliable, poor implementation
M10	3	3	1.375	1.125	2.22	3	4.72	*No*	*1*	Good date validation
M13	4	5.5	2.81	2.33	2.45	3	7.59	*No*	*3*	Well constructed
M6	0	4	3	1.86	2.14	1	7	*No*	*8*	
M01	1	3.5	2	1	1	5	4	*No*	*8*	Works, but off spec
M15	2	4	2.8	1.2	2.4	3	6.4	*No*	*9*	
M7	0	3	3	2	2	3	7	*Yes*	*10*	V.limited. Hard to test
M11	0	4	2.25	1.4	2.2	4	5.85	*Yes*	*10*	Complex, no increments, messy
M8	1	3	2.5	1.57	2.34	3.5	6.41	*Yes*	*10*	Overcomplex, incomplete, some good points
T4	4	7	2.82	2.2	2.75	3	7.77	*No*	*12*	Well implemented, robust, maintainable.
M02	1	4	2.66	1.33	1	4.5	4.99	*No*	*14*	Closer than M01, poorly commented
M17	5	6.5	2.25	2.5	1.67	3.5	6.42	*Yes*	*16*	Complex, Robust
M16	3	4.5	2.5	2.5	2.42	3	7.42	*Yes*	*16*	Good attempt.Well structured, spec problems?
T3	1	4.5	2.75	2.25	2.25	3	7.25	*Yes*	*17*	Turbo vision
M14	13	6.5	2.55	2.53	2.32	4	7.4	*Yes*	*19*	Good construction
M18	5	5.5	1.85	2.27	1.45	3.5	5.57	*No*	*20*	Quite good. Dubious HCI
M12	1	5.5	2.5	2	2.25	4	6.75	*Yes*	*45*	Well constructed

Independent Variables Shown in Italics

*See Chapter 10 for comment on values for Completeness.

Appendix 3

Questionnaire used in Chapter 4.

Prototyping in software development.

Rationale behind the questionnaire.

Little quantative data has been collected concerning the use of prototyping in software development. This questionnaire is designed to assess the extent and use of software prototyping in industry.

Allof the questions relate to the use of prototyping <u>within your organisation.</u>

Please include any comments concerning the style of the questionnaire.

Individual replies will be kept fully confidential.

Please return the questionnaire to:

Robert M. Kinmond
School of Computing
Staffordshire University
P O Box 334
FREE POST
Beaconside
Stafford
ST18 OBR

(An addressed FREE POST envelope has been provided)

Thank you for taking time to answer this questionnaire.

Robert Kinmond December 1993

(Optional)

Title | Initials | Surname

FULL COMPANY NAME

COMPANY ADDRESS

Postcode

Job Title

What is your primary job function? (Tick one box only)

- ☐ Technical customer support
- ☐ Systems programming
- ☐ Manager of IT / MIS / DP department
- ☐ Systems analysis or design
- ☐ Programming(development/ applications)
- ☐ Systems analysis and programming
- ☐ Overall Head of IT/ DP
- ☐ Networking
- ☐ Operations
- ☐ Software engineering
- ☐ Hardware engineering
- ☐ Board level Director for IT / MIS / DP
- ☐ Database administration
- ☐ Other (Please specify below)

_____

What is the primary activity of your organisation ?

- ☐ Agriculture, forestry and fishing
- ☐ Energy &water supply
- ☐ Mining, chemicals, metals & minerals
- ☐ Metal goods engineering, electronics& vehicles
- ☐ Computer manufacture
- ☐ Systems OEM
- ☐ Other manufacturing industries,Construction
- ☐ Retail, distribution, hotels, catering & repairs
- ☐ Computer dealer/reseller
- ☐ Transport & communication
- ☐ Banking, financial, business services & insurance
- ☐ Computer services bureau
- ☐ Software house
- ☐ Education, Health, government & local authorities

How many people work in the following areas ?

The whole organisation(specify) _____

Information technology (specify) _____

Which of the following System development methodologies are used within your organisation

- SSADM ☐
- YOURDON ☐
- JSD ☐
- Multiview ☐
- Information Engineering ☐
- None ☐
- Other (please specify below)

Please comment on the use of methodologies within your organisation.

I.e.: Is one methodology followed or several ?
Are the methodologies followed exactly or are they modified ?
If not indicated above which methodology is followed and why?

Which of the following techniques are used to document prototypes ?
(more than one possible)

- Data flow diagrams ☐
- Entity analysis ☐
- Entity life histories ☐
- Data dictionnairies ☐
- Systems flowcharts ☐
- Programming flowcharts ☐
- None ☐
- Other (please specify below)

Please comment on the use of prototype documentation within your organisation.

I.e.: If none is used why ?
If not indicated above which documentation technique is used and why?

To what subsequent use is the prototype put ? (more than one response possible)

- Becomes operational system (evolutionary) ☐
- System demonstration ☐
- Reusable code ☐
- Throwaway ☐
- Training ☐
- Documentation ☐
- Other (please specify below)

Please comment on the subsequent use of prototypes within your organisation.

I.e.: Does the use to which a prototype is put depend upon the application being developed, if so how ?
Are the prototypes used in any particular way, such as incremental development ?
If not indicated above what is the subsequent use of your prototypes and why?

Please indicate Prototype development times (weeks) for DSS/MIS systems.

Minimum	Average	Maximum
_____	_____	_____

Please indicate Prototype development times (weeks) for transaction processing systems.

Minimum	Average	Maximum
_____	_____	_____

Please comment on the use of prototyping for the development of TPS/MIS/DSS/etc systems within your organisation.

What are the advantages of prototyping ?

Please use the following scale
(5 = very important advantage: 1 = Very unimportant advantage)

- ☐ User involvement
- ☐ Training
- ☐ User communication
- ☐ Reduced development costs
- ☐ User satisfaction
- ☐ Reduced Maintenance costs
- ☐ Shorter development time
- ☐ Other(Please specify below)

Please comment on the advantages of using prototyping within your organisation.

e.g. If not indicated above what other advantages are gained from using prototyping and why?

What are the disadvantages of prototyping ?

Please use the following scale
(5 = very important disadvantage: 1 = Very unimportant disadvantage)

- ☐ Reduced management control of Project
- ☐ Increase in development costs
- ☐ False user expectation
- ☐ Integration within existing methodologies
- ☐ Time required for user participation.
- ☐ Inadequate analysis done
- ☐ Lack of necessary tools (Please specify below)
- ☐ Poor documentation of systems

Please comment on the disadvantages of using prototyping within your organisation.

e.g. How do you control iteration within system development ?
Other disadvantages(please specify below)

In what language are your operational systems written?

Please comment on the lanaguage used for operational systems within your organisation.

e.g. Is more than one language used, if so why ?
Is the operational system language a 3GL, 4GL, CASE tool or another technique?

In what language are your prototype systems written ?

Please comment on the language used for prototype systems within your organisation.

e.g. Is the Prototype developed using a 3GL, 4GL, CASE tool ?
Is the operational system developed in another language ?
Why/ why not ?

Thank you for taking time to answer this questionnaire.

Please enter any additional comments concerning your organisations use of prototyping in developing software systems.

e.g. How does your organisation integrate and control the process of prototyping within the system development methodology followed ?